PowerDNA User Manual

Architecture & Configuration of the Core Module for the PowerDNA Cube

June 2006 Edition
PN Man-DNA-Core-0606

Contacting United Electronic Industries

Mailing Address:
611 Naponset Street
Canton, MA 02021
U.S.A.

For a list of our distributors and partners in the US and around the world, please see http://www.ueidaq.com/partners/

Support:

Telephone:	(781) 821-2890
Fax:	(781) 821-2891

Also see the FAQs and online "Live Help" feature on our web site.

Internet Support:

Support	support@ueidaq.com
Web-Site	www.ueidaq.com
FTP Site	ftp://ftp.ueidaq.com

Table of Contents

Introduction

This document is intended to serve as a user manual to those who wish to use a PowerDNA system. It describes the PowerDNA **D**istributed **N**etwork **A**cquisition system, its components, specification, and instructs on set up and operation.

PowerDNA is the umbrella name that describes a real-time distributed I/O system with exceptional flexibility & performance. PowerDNA consists of three parts: (1) Input/Output Modules (a.k.a. I/O Modules, IOMs, Cubes) distributed throughout a process, large piece of equipment, facility, or other structure; (2) Cubes connect over copper -or- fiber optic cables to (3) a host PC with a dedicated Ethernet interface card and running Windows, Linux, or an RTOS. Optionally, cubes may be operated in stand-alone data-logger mode.

The Cube is available in either a 5- or 8-layer configuration. Two of these layers are occupied by the Core Module. The Core Module consists of the CPU Layer and the NIC (network-interface control) Layer, with connectors for either 100Base-T copper or 100Base-FX fiber-optic cable. The remaining 3 or 6 slots in the Cube are factory-configured with your selection of I/O Layers. For information on these data-acquisition layers, visit www.powerdna.com.

This document gives further details about the features and functions of various system components. Details on programming the system are contained in the companion document(s): the API Reference Manual, and layer manuals.

Who should read this manual?

This manual has been written to make the installation, configuration and operation of our PowerDNA cube as straightforward as possible. However, it assumes that the user has basic PC skills and is familiar with the Microsoft Windows XP/2000/NT/9x, QNX or Linux/RTLinux/RTAI Linux operating environments.

Organization of this manual

This PowerDNA Presentation User Manual is organized as follows:

Chapter 1—Introduction
An introduction to the cube.

Chapter 2—The PowerDNA Explorer
Provides an overview of PowerDNA Explorer Main Window, menu bar, toolbar, Device Tree, setting panel, IOM settings and Device layer settings.

Chapter 3—Three ways to communicate with IOM
Describes Synchronous and asynchronous protocols, IOM modes.

Chapter 4—Real-time operations with IOM
Describes ACB structure and function, DMap structure and function, Event notification, Heartbeat, Synchronous mode, DQE engine.

Chapter 5—Programming layer specific functions
Describes device architecture, memory map, startup sequence, setting parameters, updating firmware, Common layer interface.

Appendix
Provides an overview of how to determine the version of PowerDNA, updating the firmware, configuring the Ethernet card in various Windows OS and Linux installation.

Glossary
Alphabetical listing of key terms you will encounter in working with the PowerDNA cube and test systems in general.

Index
Alphabetical listing of the topics covered in this manual.

Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:

TIP Tips are designed to highlight quick methods to get the job done, or to reveal uncommon knowledge and ideas.

Note Notes alert you to important information.

CAUTION! Caution advises you of precautions to take to avoid injury, data loss, or a system crash.

Text formatted in **bold** typeface generally represents text that should be entered verbatim. For instance, it can represent a command, as in the following example: "Instruct operator of how to run setup using a command such as **setup.exe**"

Other PowerDNA Documentation

This *PowerDNA Manual* is one part of the documentation set available for the PowerDNA system. We offer other resources you might want to read before programming an application. They are available either on the PowerDNA Software Suite CD or can be downloaded from the UEI web site.

In particular, we recommend the *PowerDNA Programming Manual*.

Feedback

We are interested in any feedback you might have concerning our products and manuals. Comments and recommendation can be sent by email to support@ueidaq.com.

1 PowerDNA Overview

This chapter provides an overview of the key features of the PowerDNA system, and how the system works.

Thank you for purchasing a PowerDNA system. We designed this product family from the ground up to provide the best possible features, reliability and performance at an economically sound price.

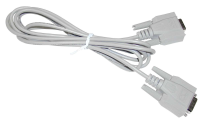
1.1 What's in the package

Inspect the package. Included you should find:

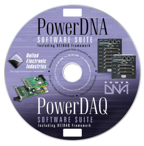
The PowerDNA Cube
Preinstalled with your selection of I/O Layers

Power supply (DNA-PSU-24: 100-240V 50-60Hz to 24VDC), accompanying daisy-chaining cable and adapter cable.

Ethernet cable with either RJ-45 connector (for copper) or SC-type (for fiber optic 100-Base-FX cable)



Serial cable (for initial configuration)



CD-ROM with support software

Additional accessories may be included, depending on your order.

1.2 Overview

The PowerDNA system consists of a hardware Cube and software suite. The software suite is located both on the PowerDNA / PowerDAQ CD shipped with the Cube and on the website: www.ueidaq.com

The software that supports the system consists of two components:

	PowerDNA Software Suite	PowerDNA low-level driver; PowerDNA Explorer (and demo); Example code for C & Java
	UEIDAQ Framework	Additional example code & docs for C/C++, C#, VB.NET, ActiveX (VB6, Delphi), MATLAB, LabVIEW, DASYLab, LabWindows/CVI, OPC

The Windows PowerDNA Software Suite contains the following software:

- *PowerDNA low-level driver*
  The interface between the cube hardware and higher-level languages.
- *PowerDNA Explorer*
  The essential tool for configuring and testing the cube. *See* Chapter 3 on use.
- *Multi-Threaded TTY Client*
  For initial setup of the cube on the network, upgrades, and calibration.
- *Example C & Java code*
  Facilitates jumping in and learning - this code will compile and execute on the cube.

In addition to the examples in the PowerDNA Software Suite, the *UEIDAQ Framework* contains example code for higher-level languages (C++, VB, Java), including graphical programming languages (e.g. LabVIEW, DASYLab).

The framework facilitates and expedites test development: an experiment can be set up in less than or twenty lines of code. The framework function calls consistent are portable between programming languages.

The Linux software package includes:

- DAQLib - Library for writing programs using PowerDNA IO modules (cubes)
- UeiPalLib - Platform abstraction library needed for building the DAQLib
- DAQLib_Samples - Example programs demonstrating how to use the DAQLib to work with various layer types

Instructions on use can be found in the readme.txt of the package.

The hardware / PowerDNA cube is composed of:

- The external casing – in two compact sizes:
  - Core Module + 3 I/O Layers: 3.95” × 4.1” × 4.0”
  - Core Module + 6 I/O Layers: 5.8” × 4.1” × 4.0”
- The Core Module [2 layers at the top]
  - The CPU Layer [PowerPC | Coldfire]
    - Integrated CPU with real-time kernel in firmware; Cube can operate as a standalone unit
  - The NIC Layer [100BaseT | Fiber 100-Base-FX]
    - Link cube to any PC over commercial Ethernet
    - Daisychain 64 Cubes over one Ethernet network
- Optional Sub-layers
  - Resolutions to 24 bits; Read/write to a Cube’s I/O Layers every 1 msec
  - Analog Input
    - High-gain & low-gain
    - Strain Gage module
    - Simultaneous Sampling module
  - Analog Output
    - with optional current/voltage booster add-in card
  - Controller Area Network Bus layer
  - Counter-Timer
  - Digital I/O
  - Power-Conversion layer

Chapter 2 details the configuration and operation of the cube’s Core Module.
Chapter 4 details the behavior and architecture of the cube’s Core Module.
Detailed information on the hardware layers is found in companion layer manuals.

2 Installation & Configuration

Installation consists of:

- PowerDNA software package installation
- Cube hardware setup
- Configuration

2.1 Initial Installation - Overview

This section outlines the steps to be taken in section 2.2.

1. Install the PowerDNA software suite. The latest software suite can be found online at www.ueidaq.com/download; a copy is also included on the CD.

2. Connect the serial cable: cube's RS-232 port -to- computer's serial port. Start a TTY client: Start ➢ Programs ➢ UEI ➢ PowerDNA ➢ MTTTY

   Change the *Baud* rate to *57600* and

   Click Connect.

3. Connect the power supply to the Cube.

4. The cube comes pre-configured with an IP address. Using MTTTY, type `[Enter]` to test the prompt, for Coldfire: `DQ>` for PPC: =>. Then type:

```
DQ> show
     ip: 192.168.100.2
netmask: 255.255.255.0
```

5. (optional) The recommended method of connection to the Cube is via a direct Ethernet cable connected to an external NIC. Connecting the cube directly to a LAN usually requires a change of IP address on the Cube. For example, your system administrator has assigned you the unused IP, 192.168.0.65. Here is how to change the IP to this example IP:

	Brief description of commands
`DQ> set ip 192.168.0.65`	Sets this Cube's IP to 192.168.1.10
`DQ> store`	Saves the newly changed configuration
`DQ> reset`	Reboots the cube for the new IP to take effect

   To make sure that the PowerDNA Cube is alive, ping it:

   ▪ `C:\> ping –n 1 192.168.0.65`

6. Use PowerDNA Explorer for graphical configuration (*see* Chapter 3).

2.2 Initial Installation – Start-to-finish Guide

This section reviews how to perform an initial hardware and software setup when you first receive a PowerDNA Cube.

2.2.1 Inspect the package

Inspect the contents of the shipping package. With a standard PowerDNA Cube you should find:

- The PowerDNA Cube itself, preinstalled with your selection of I/O Layers.
- The DNA-PSU-24 universal powerline brick, which plugs into a outlet and provides 24V dc output. The supply comes with a plug for the mains, an adapter cable ending in a Molex connector for plugging into the DNA Cube, and a daisychaining cable for supplying additional Cubes with power from the same supply (max three Cubes total).
- Serial cable for initial hardware configuration and firmware downloading.
- CD-ROM with support software

2.2.2 Install Software

This section describes how to load the PowerDNA software suite onto a Windows- or Linux-based computer and run some initial tests.

The latest PowerDNA support software is online at www.ueidaq.com/download; a known working copy is also on the PowerDNA Software Suite CD.

Software Install: Windows 9x/2000/XP

The PowerDNA CD provides two installers:

- PowerDNA Software Suite: low-level driver and PowerDNA tools
- UEIDAQ Framework: high-level programming examples (optional)

Both installers automatically search for third-party IDE and testing suites, and add themselves as tools to the found suites. Install third-party applications (e.g. LabVIEW, MsVS2003) before the PowerDNA Software Suite or UEIDAQ Framework.

Install the PowerDNA Software Suite:

1. Log in as Administrator.
2. Run Setup
   a. Insert the PowerDNA Software Suite CD into your CD-ROM drive. Windows should automatically start the PowerDNA Setup program. An installer with the UEI logo and then PowerDNA Welcome screen should appear. If none appears, run **setup.exe** from the CD drive: Start ➢ Run ➢ **d:\setup.exe** ➢ OK.
   b. If you downloaded the most recent executable from the www.ueidaq.com, double-click to run the executable.
2. Choose the **PowerDNA Software Suite** option
3. Unless you are an expert user and have specific requirements, we advise you to select **Typical** installation and accept the default configuration. The Software Suite installer requires and automatically installs Sun's Java VM (JRE) for you, in addition to the full compliment of tools. Alternatively, use the custom Custom option to display and ensure that all of the packages necessary are installed.
   - Companion Documentation:
     Quick Start Guide, Configuration & Core Module,
     I/O Layer Manuals, Low-level Programming Guide
   - SDK: includes/lib for C/java, examples, and Sun's JRE;
     The SDK is not the UeiDaq Framework.
   - PowerDNA Apps: PowerDNA Explorer, MTTTY
   - PowerDNA Components (incl. DLL files)
   - PowerDNA Firmware
4. Click **Next** to continue through the dialogues.
5. Click **Finish** to complete installation; restart the computer.

This Software Suite installed the bare-minimum tools needed in later steps: MTTTY, PowerDNA Explorer, and the low-level driver.

The UEIDAQ Framework provides the scaffold for developing applications under C/C++, C#, VB.NET, ActiveX (VB6, Delphi), MATLAB, LabVIEW, DASYLab, LabWindows/CVI, OPC, and other programming languages.

Note Because the installation process modifies your Windows registry, you should always install or uninstall the software using the appropriate utilities. Never remove PowerDNA software from your PC directly by deleting individual files; always use the Windows Control Panel/Add-Remove Programs utility.

Software Install: Linux
Linux: The PowerDNA_*.tgz file in the CD\Linux folder contains the software package for Linux. To extract the file to a local directory:

```
  tar -xjvf /path/to/powerdna*.tgz
```

Follow the instructions in the readme.txt contained therein.

2.2.3 Initial Boot-up (prep. for network configuration)

1. Familiarize yourself with front-panel layout (note: all connections are made on front of the unit; no rear access is required in rack-mounted configuration).

2. Attach the serial cable to the host PC and to the DNA Cube's RS-232 port.

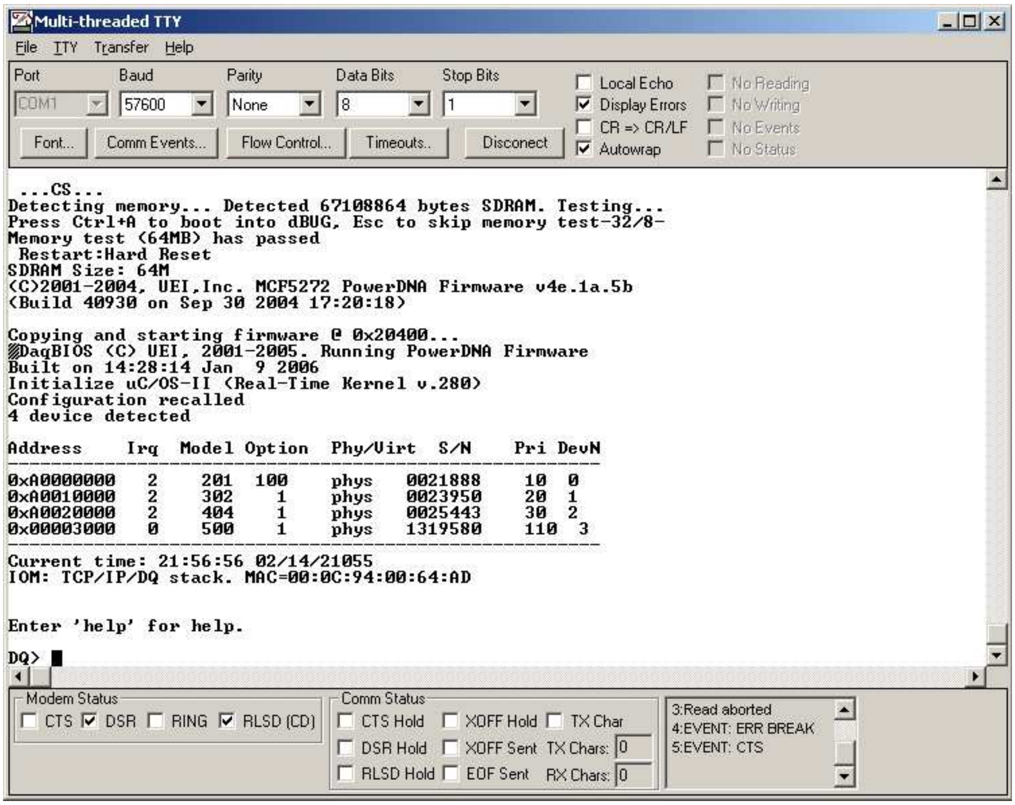
Run a terminal-emulation program (MTTTY) on the PC. Any terminal-emulation program may be used (MTTTY, Minicom, TeraTerm, etc.) Note that HyperTerminal will most likely not work with a PowerDNA Cube.

Verify that COM parameters are set: 57600 baud, 8 bits, no parity, 1 Stop bit.

Click Connect in MTTTY, or use the commands on one of the other terminal-emulation programs to establish communications with the Cube.

3. Power up the Cube (9-36V DC) by attaching the Molex-type power connector leading from the bundled DNA-PSU-24, a user-supplied source, or a daisychained line from another PowerDNA Cube. Note that the DNA-PSU-24 plugs into a 100-240V, 50/60-Hz outlet. Note that the Cube does not have an On/Off switch.

4. As soon as the Cube powers up, it runs through self-diagnostic mode and generates output on the terminal program. A typical readout might be:



Port: COM1 Baud: 57600 Parity: None Data Bits: 8 Stop Bits: 1

Local Echo [✓ Display Errors] CR => CR/LF [✓ Autowrap]

```
...CS...
Detecting memory... Detected 67108864 bytes SDRAM. Testing...
Press Ctrl+A to boot into dBUG, Esc to skip memory test-32/8-
Memory test (64MB) has passed
 Restart:Hard Reset
SDRAM Size: 64M
(C)2001-2004, UEI,Inc. MCF5272 PowerDNA Firmware v4e.1a.5b
(Build 40930 on Sep 30 2004 17:20:18)

Copying and starting firmware @ 0x20400...
DaqBIOS (C) UEI, 2001-2005. Running PowerDNA Firmware
Built on 14:28:14 Jan  9 2006
Initialize uC/OS-II (Real-Time Kernel v.280)
Configuration recalled
4 device detected

Address     Irq  Model Option  Phy/Virt  S/N     Pri DevN
---------------------------------------------------------
0xA0000000   2    201   100    phys    0021888    10  0
0xA0010000   2    302     1    phys    0023950    20  1
0xA0020000   2    404     1    phys    0025443    30  2
0x00003000   0    500     1    phys    1319580    110  3
---------------------------------------------------------
Current time: 21:56:56 02/14/21055
IOM: TCP/IP/DQ stack. MAC=00:0C:94:00:64:AD


Enter 'help' for help.

DQ> 
```

Modem Status: CTS [✓ DSR] RING [✓ RLSD (CD)]

Comm Status: CTS Hold, XOFF Hold, TX Char, DSR Hold, XOFF Sent, TX Chars: 0, RLSD Hold, EOF Sent, RX Chars: 0

3:Read aborted
4:EVENT: ERR BREAK
5:EVENT: CTS

The boot process displays the model, serial number, and position of layers. Type **show <CR>** to display information on cube configuration:

```
DQ> show
        name: "IOM_1234"
       model: 0x1005
      serial: 0012345
         mac: 00:00:11:AA:BB:CC
        fwct: 1.2.0.0
         srv: 192.168.100.3
          ip: 192.168.100.2
     gateway: 192.168.100.1
     netmask: 255.255.255.0
         udp: 6334
```

Brief description of show command parameters
IOM or I/O Module – is another name for a Cube
Core Module > Model Number (1005: ColdFire)
Core Module > Serial Number (S/N) of Cube
Core Module > NIC Layer > MAC Address
Define Cube procedure on startup
IP Address of firmware server
IP Address of this Cube
IP Address of gateway
IP Subnet Mask of this Cube
UDP Port to receive commands on

All parameters can be changed; most notably, the cube's configured IP, gateway, and subnet mask (netmask).

2.2.4 IP Addresses on the PowerDNA Cube

The PowerDNA Cube ships with a preconfigured factory default IP address in nonvolatile memory (generally 192.168.100.2). This is a static IP address; the PowerDNA Cube never retrieves its IP address from a DHCP server.
This section describes why and how to change the default IP address.

Should you change the IP?

Yes, if you plan to use the Cube on a LAN where..

- high sampling rate is not necessary
- some samples can be dropped due to network congestion and collisions
- the cube should be accessible by multiple parties on the LAN
- multiple Cubes operate (and interact) on the same network

Alternatively, if you plan to use the Cube for high-speed measurements where reliability is necessary – a direct connection between the host PC and a NIC[1] is recommended. For a direct connection, see the following section, "Improving Network Performance"

How to change the IP:

Both PowerDNA Explorer and a terminal-emulation program can change the IP. Consult your system or network administrator to obtain an unused IP. Let's say, for example, that your system administrator assigns you the IP 192.168.0.65.

To change the IP from the terminal program:

```
DQ> set ip 192.168.0.65
Enter user password > powerdna
DQ> store
DQ> reset
```

Sets this Cube's IP to 192.168.0.65
The default password is "powerdna"
Saves the newly changed configuration
Reboots the cube for the new IP to take effect

You can set any parameters listed with the "show" command in this manner.

Connect the PowerDNA Cube to your switch with the bundled CAT5e cable.

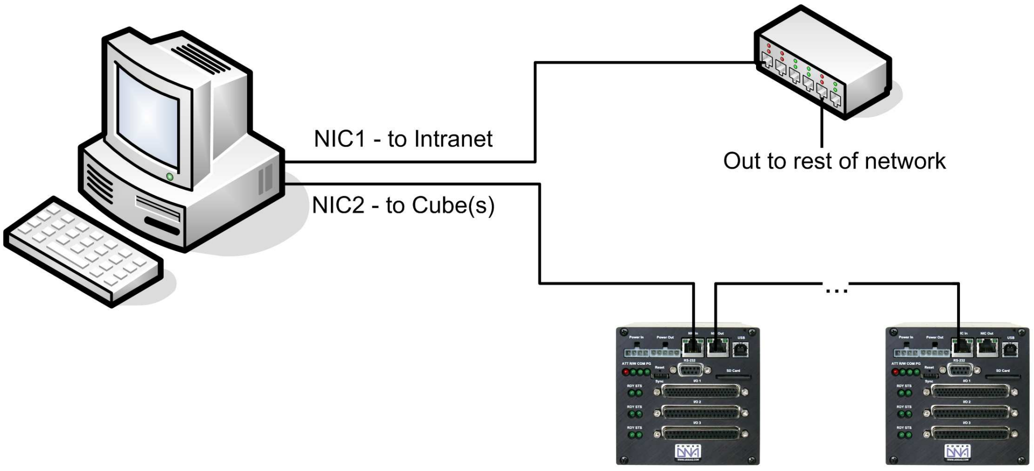
If you can establish communications with a Cube but later want to modify the IP address, you can also do so from within PowerDNA Explorer. After the exploratory process, go to the field where the application displays the IP address. You enter the new IP address and then hit <Return>. This action downloads the new IP address into the Cube's non-volatile memory. You might also need to change the gateway and network mask to match settings on your LAN.

[1] NIC - Network Interface Controller; a commercially available Ethernet (i.e. IEEE 802.3) adapter.

2.2.5 Improving Network Performance

To improve PowerDNA network performance, we recommend that you use a separate commercially available network interface controller (NIC) card and set up a dedicated mini-network for PowerDNA.

The goal of this section is to facilitate creation of such a network:



For example, assume that your office uses a Class C network (the class intended for small networks with fewer than 256 devices) and your host is configured with a static IP or via DHCP—Dynamic Host Configuration Protocol—a protocol for assigning dynamic IP addresses to devices on a network.

1. Obtain your networking configuration by using the Command Prompt:

- Start ➢ Programs ➢ (Accessories ➢) Command Prompt
- `C:\> ipconfig`

```
Ethernet adapter NIC1 - Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.1.10
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.1
```

Linux users can use the more verbose “`ifconfig`” command instead.

Here, the subnet range 192.168.1.0-192.168.1.255 is being used by NIC1.

IP Addressing:
The range of usable addresses is defined by the IP address and subnet mask. An IP address is a number that is split into the range of 0.0.0.0 and 255.255.255.255. Here, the IP address is 192.168.1.10.
The subnet mask indicates where an address stops. For example, a subnet mask 255.255.255.240 has 15 usable addresses (255.255.255.255 – 255.255.255.240).
Here, the subnet is 255.255.255.0, or 255 addresses.
The subnet limits from anything.anything.anything.0 up to the max.
The usable range for 192.168.1.10/255.255.255.0 is 192.168.1.1 to 192.168.1.254 (192.168.1.0 and 192.168.1.255 are reserved for Router and Broadcast messages).
The usable range for 192.168.0.4/255.255.0.0 is 192.168.0.1 to 192.168.255.255
The usable range for 192.168.100.2/255.255.255.0 is 192.168.100.1 to 192.168.100.254

Not every IP address from 0.0.0.0 to 255.255.255.255 is usable; however, these three ranges of IP addresses are guaranteed open for private use:

10.0.0.0 – 10.255.255.255
172.16.0.0 – 172.31.255.255
192.168.0.0 – 192.168.255.255

One need not use the entire set.

2. Install the secondary NIC card.

3. Set up a network that does not overlap the existing one.
   The address space 192.168.1.0-192.168.1.255 is used. The IP address block, 192.168.2.1-192.168.2.255 is available and in the private range.

   Let us choose 192.168.100.1-192.168.100.255 for the PC's secondary NIC:

IP:	192.168.100.3
Netmask:	255.255.255.0
Gateway:	192.168.100.3

   Using the Network (Connections) in the control panel:
   Start ➢ Programs ➢ Control Panel ➢ Network (Connections)
   Right-click the adapter to bring up the properties window.
   Open the TCP/IP properties of the adapter and edit to your liking.

   See the Appendix at the end of this document: "Configuring a Second Ethernet Card" for step-by-step instructions on how to do this.

4. Confirm the network configuration at the Command Prompt:

- Start ➢ Programs ➢ (Accessories ➢) Command Prompt
- `C:\> ipconfig`

```
Ethernet adapter NIC1 - Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.1.10
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.1

Ethernet adapter NIC2 - Local Area Connection 2:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.100.3
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.100.3
```

5. Set up the PowerDNA Cube to use the same subnet, namely:

Cube IP:	192.168.100.2	*this is the factory default*
Gateway:	192.168.100.3	
Netmask:	255.255.255.0	

To do this from a serial terminal-emulation program, when you get the DQ command prompt enter the following commands:

```
DQ> set ip 192.168.100.2
DQ> set gateway 192.168.100.3
DQ> set netmask 255.255.255.0
DQ> store
DQ> reset
```

Brief description of commands

Sets this Cube's IP address to 192.168.100.2
Sets this Cube's Gateway to 192.168.100.3
Sets the subnet mask to 255.255.255.0
Saves the newly changed configuration
Reboots the cube for the new IP to take effect

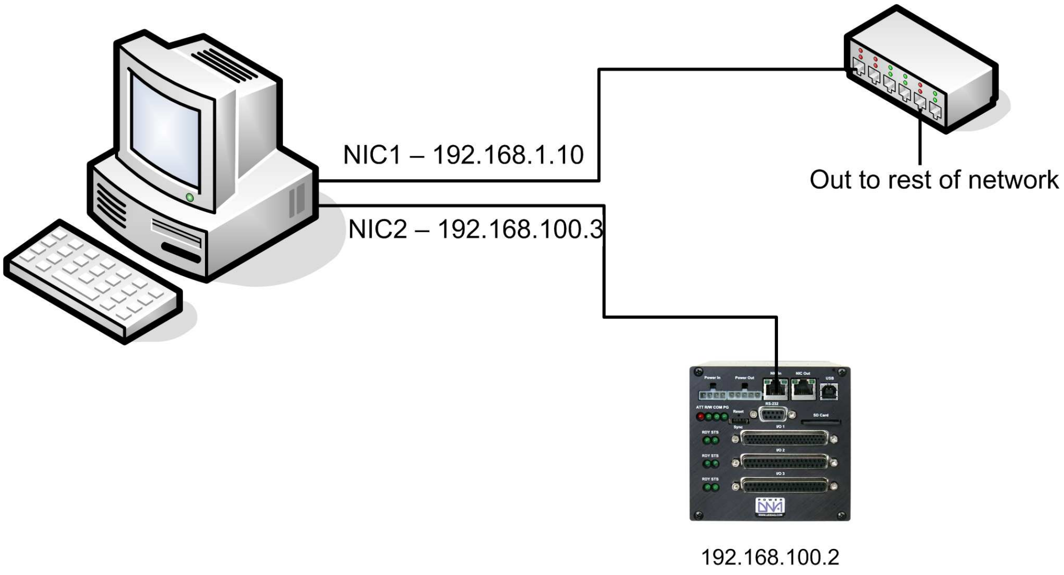
6. Connect the PowerDNA Cube to your PC's second NIC using the bundled CAT5 cable. The green lights should light up (try the other port, otherwise).

7. Ping the cube to make sure that it is alive.
   - `C:\> ping –n 1 192.168.100.2`

```
Pinging 192.168.100.2 with 32 bytes of data:

Reply from 192.168.100.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.100.2:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
```

The above is a successful response. A `Request timed out` message indicates error.

7. The cube should now be configured as follows:



8. You may now use PowerDNA Explorer to access the cube. *See* Chapter 3.

Troubleshooting

The following checklist may assist you in troubleshooting a Cube.

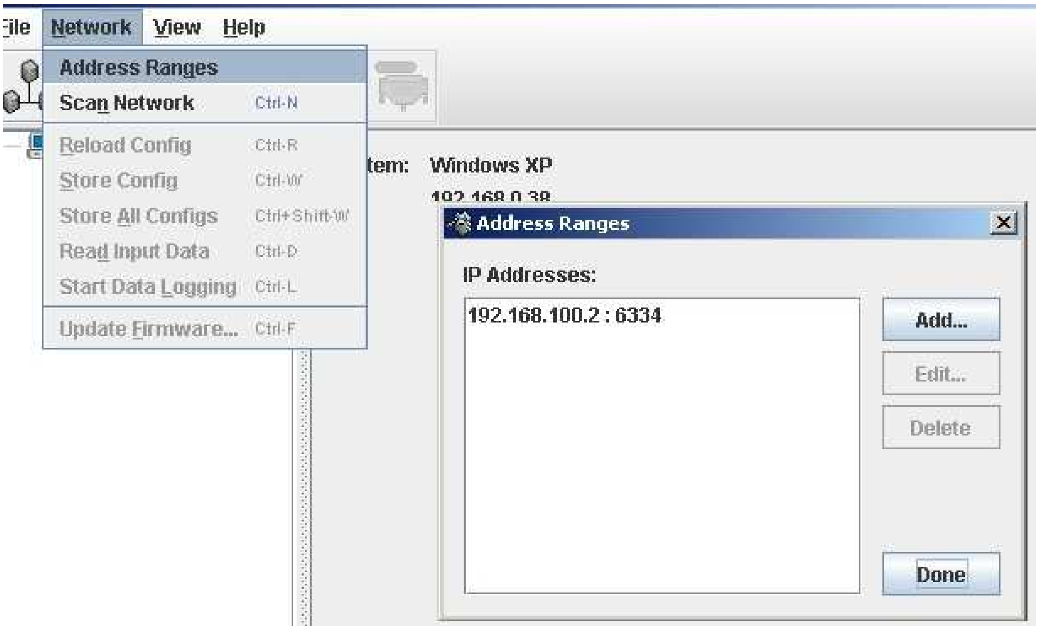
✓ The PG (Power Good) LED is on: the cube plugged in using 9-36V DC.
✓ The green lights on NIC In or NIC Out are blinking: CAT5e cable connected
✓ Use the command prompt to `ping <cube IP>` (e.g. `ping 192.168.100.2`)
- Disable (temporarily) the firewall on the secondary NIC
- Check the secondary NIC's network settings
- Check the cube's network settings
  - Use MTTTY and hit Connect
  - Press `[Enter]` to display the `DQ>` or `=>` prompt.
    No prompt indicates that you are not connected:
    - Verify that the serial cable is firmly connected to the RS-232 port
    - Verify the settings: 57600 baud, no parity, 8 data bits, 1 stop bit
    - Try COM1, COM2, COM3 then hit Connect and press `[Enter]`
      - Reboot the cube. The start-up screen should display upon restart.
        - If all else fails, contact UEI support at: *support@ueidaq.com*
  - Type "`show`" to verify the Cube's IP, Subnet Mask, and Gateway
  - Ensure that the computers are on a valid subnet and have valid IPs
    - Contact UEI support at: *support@ueidaq.com*

2.2.6 PowerDNA Explorer Quick-start

PowerDNA Explorer does just what its name implies: it “explores” the LAN, looking for connected PowerDNA Cubes. Chapter 3 covers the PowerDNA Explorer in detail. This section/page only provides a quick-start guide.

The PowerDNA Explorer identifies PowerDNA Cubes on a selected network – the discovered Cubes are listed on the left-hand-side pane. Select a cube to display pertinent hardware and firmware information. Select a layer of a specific cube to manipulate its inputs or outputs. In brief, this useful tool lets you verify that the Cube is communicating with the host, and that the I/O Layers are functioning properly.

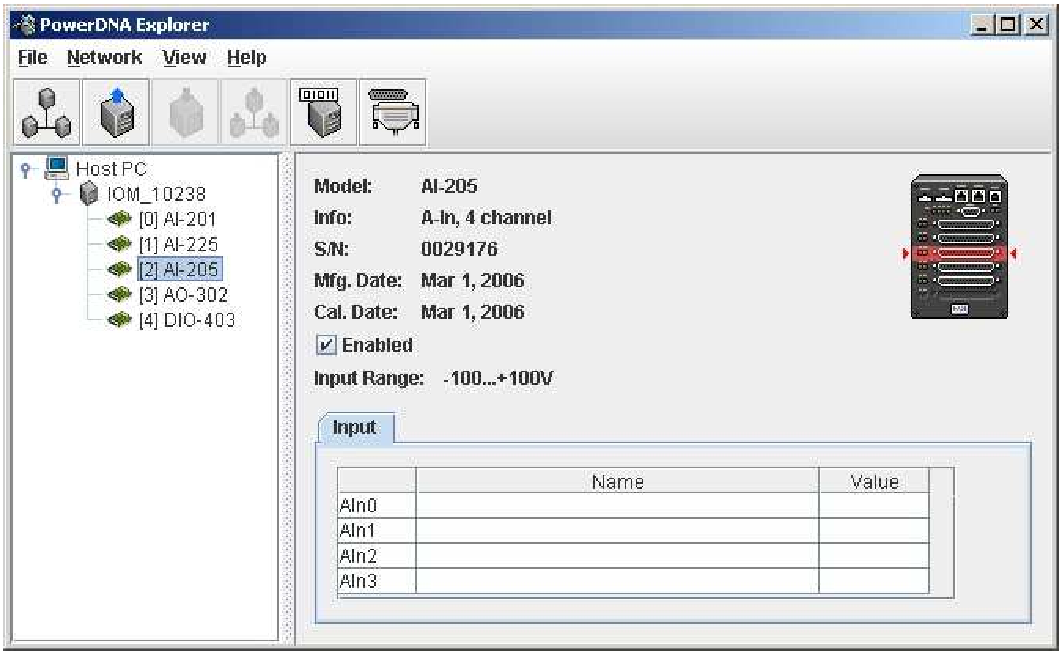
To scan the network for PowerDNA Cubes, provide a set of addresses to scan: Select Network ➢ Address Ranges from the menu:



Add the IP address of the PowerDNA Cube (e.g. 192.168.100.2); click Done. Now scan the LAN for PowerDNA Cubes: Network ➢ Scan Network

One or more gray cube-like icons will display in the left-hand-side of the cube. If no cube icons are displayed, see the Troubleshooting note in the previous section.

Double-click a cube to see its information and list the layers:



Model: AI-205
Info: A-In, 4 channel
S/N: 0029176
Mfg. Date: Mar 1, 2006
Cal. Date: Mar 1, 2006
✓ Enabled
Input Range: -100...+100V

	Name	Value
AIn0		
AIn1		
AIn2		
AIn3		

The above screenshot is from the PowerDNA Explorer Demo. The "demo" is just a simulator for users without cubes – or for new users that want to explore the PowerDNA Explorer program without reading/writing to real hardware. Run this program, and hover your mouse over the buttons to read the tool-tips and learn through interacting with the program.

Some quick notes:

- ✓ To use the layer, the "Enabled" check box should be set.
- ✓ To read from a layer, click the second-to-last button: "Read Input Data"
- ✓ To write to the layer, change the value and click the third (or fourth) button with the red arrow on top of the cube: "Store Configuration". The cube with the blue arrow above it restores the configuration.
- ✓ To change the IP, change the number, unselect the field, and "Store Configuration". Take care not to set the IP Address to outside of the network's configuration subnet -or- to an IP address that is currently in use as the cube will become unreachable.

See chapter 3, PowerDNA Explorer, for additional information and instruction.

2.2.7 Updating Firmware

Firmware in a PowerDNA Cube's CPU layer stores configuration data, along with a user application (user-app is compiled on a host PC).

Updated firmware is regularly released to introduce new features, and to improve the performance of existing features. Updated releases of the firmware are bundled with the entire PowerDNA Software Suite, available for download at any time from the UEI web site (www.ueidaq.com).

CAUTION!
If you update the firmware in a Cube, be sure to use the PDNA Explorer from the same release as that new firmware.

After installing the PowerDNA Software Suite, browse to the installation's Firmware directory (e.g. *C:\Program Files\UEI\PowerDNA\Firmware*).

The directory may contain MTTTY, updated firmware installation instructions "FirmwareInstall.html," and two sub-directories containing the firmware. Choose the sub-directory corresponding to the architecture of your cube: ColdFire (CF/CM) with extension S19, or PowerPC (PPC), with extension MOT.

Determining the version of your PowerDNA cube with PDNA Explorer

Before updating the firmware of a PowerDNA cube, check the cube's version to determine which update method to use:

1. Supply power to the PowerDNA cube.

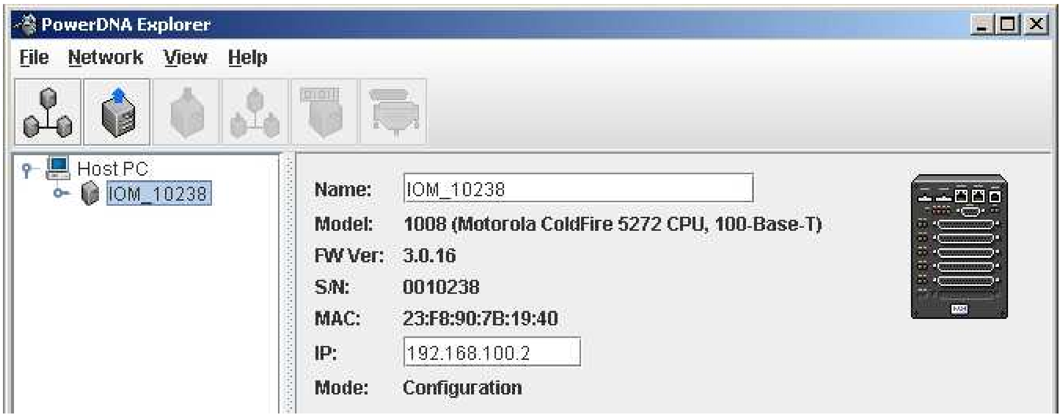
2. Connect the PowerDNA cube to its network.

3. Start PowerDNA Explorer on the Microsoft Windows desktop from
Start ➢ Programs ➢ UEI ➢ PowerDNA ➢ PowerDNA Explorer

4. Choose Network ➢ Scan Network

5. Select on the PowerDNA cube you wish to query (by clicking the cube).

6. The version is given in the **FW Ver** field.



Name: IOM_10238
Model: 1008 (Motorola ColdFire 5272 CPU, 100-Base-T)
FW Ver: 3.0.16
S/N: 0010238
MAC: 23:F8:90:7B:19:40
IP: 192.168.100.2
Mode: Configuration

If the **FW Ver** field has is version 2.x.x, or 3.x.x (let x be any version number), then you should follow the *Firmware update instructions [CM5, CM8]* section below. For other versions of firmware (e.g. 1.x.x), use the manual on the CD that accompanied your device when you purchased it.

2.2.8 Firmware update instructions

Before using a new release of the libraries and applications to communicate with your PowerDNA cube, you must install the latest version of the firmware onto the PowerDNA cube. The version of the firmware must correspond to the version of the PowerDNA Software Suite; mismatched versions cause an error.

Instructions for updating the PowerDNA Cube via PowerDNA Explorer (over Ethernet LAN line), and over MTTTY (serial line) follow.

To upload firmware with PowerDNA Explorer over LAN:

1. Supply power to the PowerDNA cube.

2. Connect the PowerDNA cube to its network.

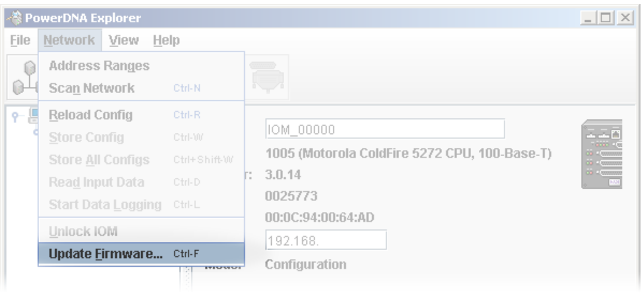
3. Start PowerDNA Explorer on the Microsoft Windows desktop from

Start ➢ Programs ➢ UEI ➢ PowerDNA ➢ PowerDNA Explorer

4. Choose Network ➢ Scan Network

5. Select the PowerDNA cube to update.

6. Select *Network→Update Firmware…*from the menu.

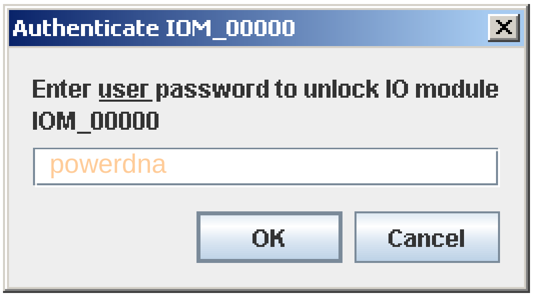


Update Firmware menu item

7. Click on "Yes" when you are prompted, "Are you sure you want to update firmware…"

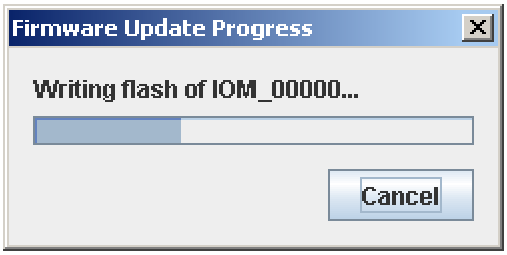
8. Double-click on the dq_ram.S19 file.

9. Enter the password to continue. More information about passwords can be found in the *Interfacing to the CM module using a serial interface* section of this manual. PowerDNA cubes come with the default password set to **`powerdna`**.



Password dialog box

10. Wait for the progress dialog to complete, then the PowerDNA cube will be updated and running the new firmware.



Firmware Update Progress dialog box

Each cube is updated in three steps. First, the firmware is transferred to the cube. Second, the firmware is written to the flash memory. During this step, the R/W light on the front of the cube is lit, in addition to the PG light. Third, the cube is reset. During this step, the ATT, COM, and PG lights are lit, and the R/W light will turn on and off periodically. When the cube is finished resetting only the PG light is lit.

To upload firmware over serial port using a terminal client (MTTTY):

Under DNA-CM5 and DNA-CM8:

1. Establish communications between the PC and a Cube over the serial link.

2. Use the hardware Reset switch on the front of the Cube to reset the CPU Layer.

3. While the Cube is starting up again, `Press <Ctrl>+<A>` to activate the download screen (indicated by a #> prompt).
If you get to the `DQ>` prompt, you waited too long and must return to Step 2.

4. Enter the **`dl`** command to enter the firmware-download routine.

5. Transfer the file. Depending on which terminal-emulation package you decide to run, you generally initiate the download with a command similar to “send”. In MTTTY, go to the top menu and select Transfer->Send file (text). When it asks for a file, go to the PowerDNA\Firmware directory and select the `.S19` or `.MOT` firmware file. The download procedure will take roughly a minute.

6. To tell the Cube to save the new firmware into EPROM, enter the commands

```
upuser <CR>
update
```

7. Enter **`go`** to complete the firmware-update procedure and return to the DQ> prompt.

Under DNA-PPC5 and DNA-PPC8:

1. Establish communications between the PC and a Cube over the serial link.

2. Use the hardware Reset switch on the front of the Cube to reset the CPU Layer, or type: **`reset all`**

3. While the Cube is starting up again, `Press` **`ESC`** to go into u_boot.

4. Type the command to erase firmware download area in the Flash memory:

```
=> erase all
=> loads romimage.mot
```

loads stores firmware into the flash while downloading it.

5. Transfer the **mot**orola firmware file. Use Transfer » Send File, and select \Program Files\UEI\PowerDNA\ Firmware_PPC\romimage_3_x_y.mot
A progress bar will appear in the lower left corner of MTTTY indicating progress.

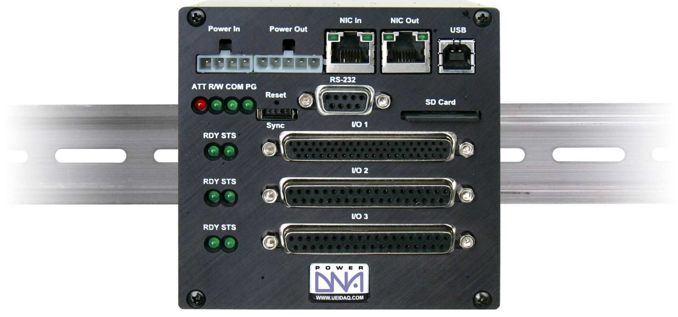
6. Wait for the upload to complete (it may take a few minutes).

7. After the process finishes, enter the **fwjmp** command, then the PowerDNA cube will be updated and running the new firmware. At this point, only the PG light on the cube remains lit.

2.3 Mounting and field connections

Mount the Cube directly to the application hardware either by screwing it directly to the machine or by using the optional DIN rail clip (DNA-DR). A normal DIN rail comes with screws with which mount the rail onto another surface or piece of equipment. However, because the Cubes are designed to fit into applications where space is at a premium, it might sometimes be difficult to attach the rail in this way. For such cases, we include a special adhesive tape for attaching the rail to any desired surface.

CAUTION! Take care when deciding on which surface you plan to mount a Cube. For example, with the adhesive strip you can normally attach the DIN rail to a wall without causing any damage—unless the wall has a sensitive coating such as delicate paint or wallpaper..



2.4 PowerDNA Cube Wiring

Refer to the companion layer manuals for proper wiring to layers.

2.5 Repairing (and upgrading) Your Cube

PowerDNA Cubes come the factory fully configured and calibrated. They are not suited for field upgrades or repairs. Should you encounter a problem with a Cube, or should you want to enhance or otherwise modify the selection of I/O layers in a Cube, you must send the unit back to the factory or to your local distributor. This process requires that you request an RMA number from UEI. To do so, you must provide the following information:

1. Model Number of the Cube
2. Serial Number of the Cube
3. Reason for return
   - Calibrating the layer(s)
   - Defective layer for repair
   - Upgrade with additional layer(s)

UEI will process the request and issue an RMA along with an estimate on the work involved to handle your request as well as the associated costs.

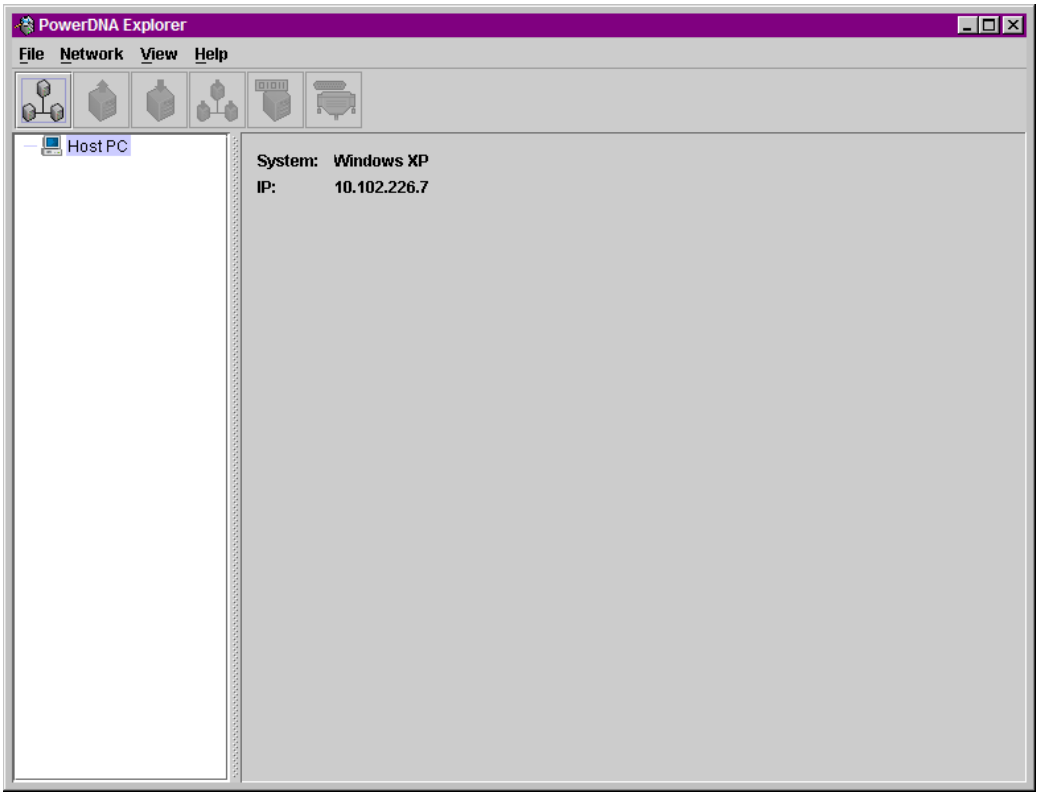
3 PowerDNA Explorer

PowerDNA Explorer simplifies configuration and setup of a PowerDNA cube under Microsoft Windows.

This section describes the different menus in PowerDNA Explorer.

Note There is a PowerDNA Explorer Demo; the demo lets a user safely explore the menus and layer screens without the need for actual PowerDNA cubes.

3.1 The Main Window



System: Windows XP
IP: 10.102.226.7

Main Window

The main window is the window that the user sees when the PowerDNA Explorer is first launched, and is where the user does most of his work. It has four main parts: the menu bar, the toolbar, the device tree, and the settings panel.

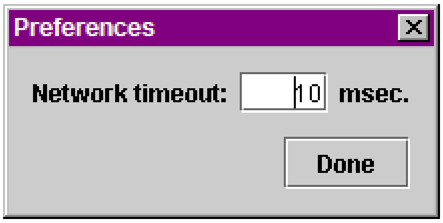
3.2 Menu Bar

The menu bar contains the following menus and menu items.

3.2.1 File Menu

Preferences brings up the preferences dialog.

The preferences dialog allows the user to specify the network timeout interval. This is the length of time PowerDNA Explorer will wait for response from a PowerDNA cube before giving up with an error. It defaults to 100 milliseconds.



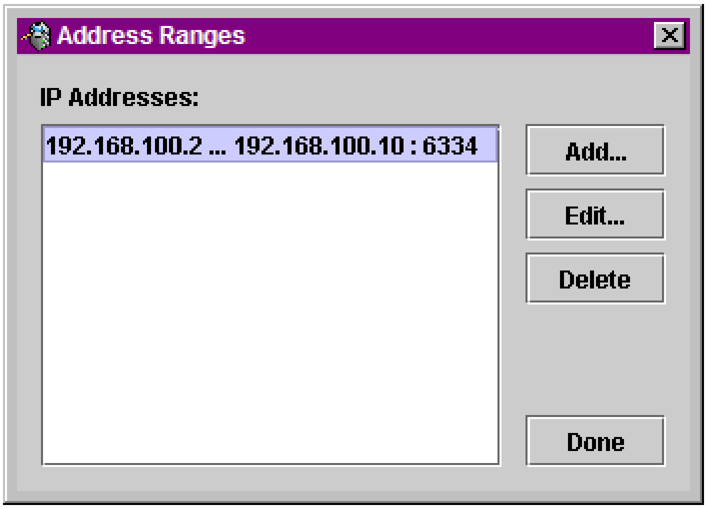
Network timeout: 10 msec.

Preferences

Exit exits the application. If there are unsaved device settings changes, the user is prompted for confirmation.

3.2.2 Network Menu

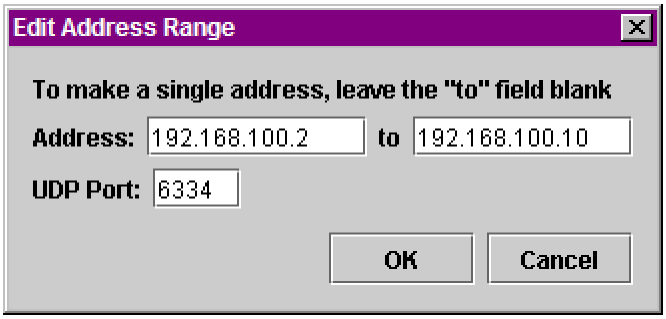
Address Ranges brings up the Address Ranges dialog, allowing the user to specify where to scan for devices.


Address Ranges

IP Addresses:

192.168.100.2 ... 192.168.100.10 : 6334

Address Ranges dialog box


Edit Address Range

To make a single address, leave the "to" field blank

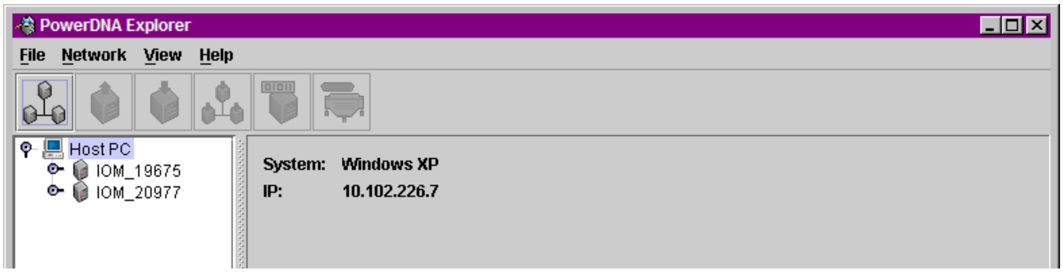
Address: 192.168.100.2 to 192.168.100.10

UDP Port: 6334

Edit Address Range dialog box

The Address Ranges dialog allows the user to specify the IP addresses and UDP port to use to find devices. The user can specify individual addresses, as well as address ranges. The specified items appear in a list that can be added to and deleted from. This list defaults to containing a single range item which specifies the range 192.168.100.2 ... 192.168.100.10.

Scan Network scans the network for devices, and populates the device tree. How much of the network is scanned depends on the settings in the Network Ranges dialog.



System: Windows XP
IP: 10.102.226.7

After a Network→Scan Network

If the user chooses Scan Network when the device tree is already populated, any new devices discovered will be added to the tree. Any existing devices that are missing will be removed from the tree, unless the user has made unsaved changes to such a device's configuration, in which case it will be marked in the tree as missing.

Reload Config re-reads the configuration of the current device selected in the Device Tree. If the user has made changes to the settings in the settings panel for the current device, Read will replace those settings with the device's current settings, after prompting the user for confirmation.

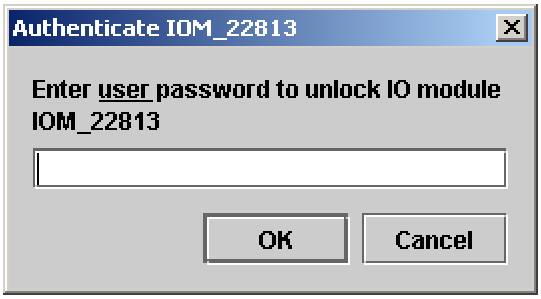
Store Config writes the currently selected device's changed settings to the device. The button is disabled for devices that haven't been modified.

Store All Configs writes all of the changed devices' settings to the devices. The button is disabled if no devices have been modified.

Read Input Data is enabled when the currently selected device is an input device layer. It reads the current input values to the device and causes them to be displayed in the settings panel.

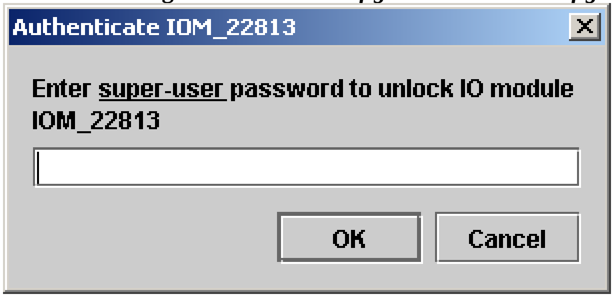
Update Firmware… loads a firmware update file to all connected PowerDNA cubes if Host PC is selected. It updates only one PowerDNA cube when the specific PowerDNA cube is updated. More details about this can be found in the section *Updating the firmware in a version 2.0 PowerDNA cube*.

Note that writing certain configuration changes to a PowerDNA cube running firmware 2.0.16 will bring up a password dialog box. More information about passwords can be found in the *Interfacing to the CM module using a serial* *interface* section of this manual. PowerDNA cubes come with the default password set to “powerdna”.


Authenticate IOM_22813

Enter user password to unlock IO module IOM_22813

Password dialog box for *Store Config* and *Store All Configs*

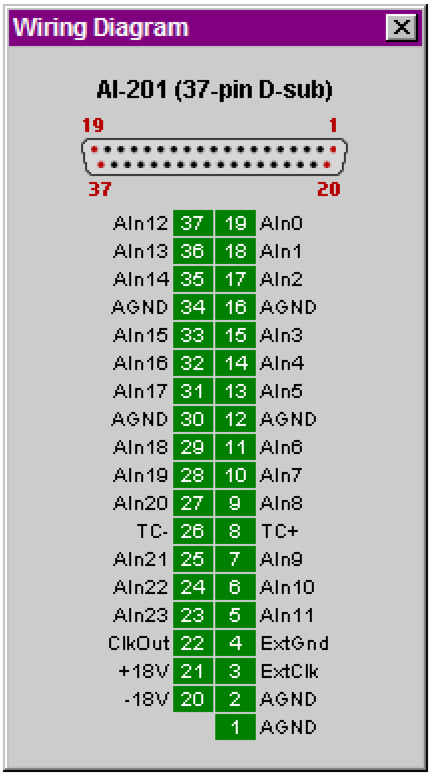

Authenticate IOM_22813

Enter super-user password to unlock IO module IOM_22813

Password dialog box for *Update Firmware...*

3.2.3 View Menu

Show Wiring Diagram is a friendly reminder of the connector pins for a specific layer. All layers have this option, and we display this one as an example. The wiring diagrams in PowerDNA Explorer match the wiring diagrams in this manual in the sections for each layer.



Wiring Diagram

AI-201 (37-pin D-sub)

AIn12	37	19	AIn0
AIn13	36	18	AIn1
AIn14	35	17	AIn2
AGND	34	16	AGND
AIn15	33	15	AIn3
AIn16	32	14	AIn4
AIn17	31	13	AIn5
AGND	30	12	AGND
AIn18	29	11	AIn6
AIn19	28	10	AIn7
AIn20	27	9	AIn8
TC-	26	8	TC+
AIn21	25	7	AIn9
AIn22	24	6	AIn10
AIn23	23	5	AIn11
ClkOut	22	4	ExtGnd
+18V	21	3	ExtClk
-18V	20	2	AGND
		1	AGND

Example of a Wiring Diagram

3.2.4 Help Menu

About PowerDNA Explorer shows the about box, which shows the program icon, program name, version number, company name, and copyright notice.

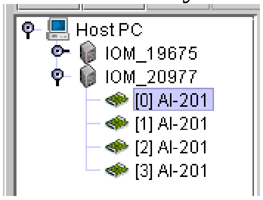
3.3 Toolbar

The toolbar contains the following buttons: Scan Network, Reload Config, Store Config, Store All Configs, Read Input Data, and Show Wiring Diagram. They duplicate the functionality of the corresponding menu items as described above.

3.4 Device Tree

When the application is first launched, the tree contains just a root item representing the host computer. When the user selects Scan Network from the Network menu or the toolbar, the device tree gets populated with all central controllers, IOMs, and device layers accessible from the network, as filtered through the Network Ranges dialog. Central controllers, if any, appear as children of the Host PC item. IOMs that are connected to the PC without use of a central controller also appear as direct children of the Host PC item.

Each item has an icon indicating whether it is a central controller, IOM, or layer. The text label for each item is the device's model number, name, and serial number. Layers are also labeled with their layer number in parentheses.



Example device tree

When an item is selected in the tree, the settings panel changes to reflect the settings for that device. The first time an item is selected, the device is queried as though the user had invoked the Read command. On subsequent selections of the same item, the last settings are re-displayed. Thus, if the user made changes but did not write them to the device, the changes are remembered. Invoking the Read command will re-read the device and overwrite the current settings in the settings panel.

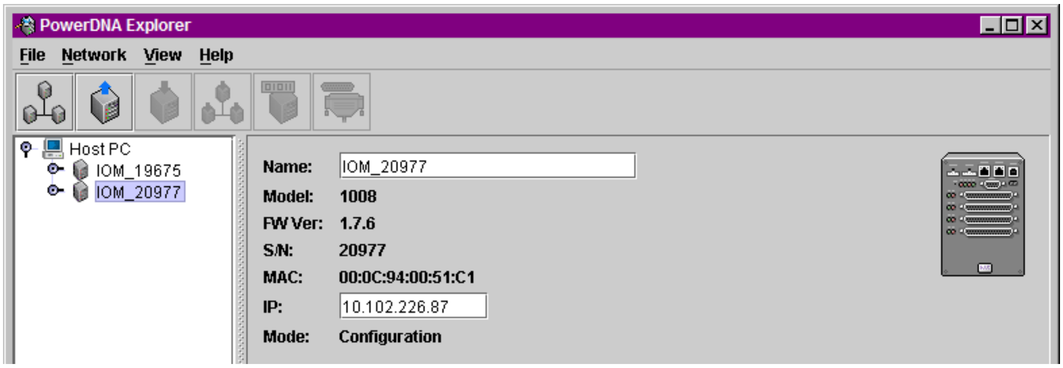
Devices whose settings have changed but not been written are displayed in bold italics in the tree to provide a visual cue to the user. Changed devices that become missing on a subsequent invocation of Scan Network turn red in the tree. (Unchanged items that become missing are simply removed from the tree.)

3.5 Settings Panel

The settings panel presents a set of controls allowing the user to change the settings of the current device selected in the device tree.

3.5.1 IOM Settings

The settings panel has the following controls when an IOM is selected in the tree.



Name: IOM_20977
Model: 1008
FW Ver: 1.7.6
S/N: 20977
MAC: 00:0C:94:00:51:C1
IP: 10.102.226.87
Mode: Configuration

Example IOM Settings panel for a PowerDNA cube

Name shows the IOM name. It can be changed.

Model shows the IOM's model number.

FW Ver shows the version of the firmware installed on the PowerDNA cube.

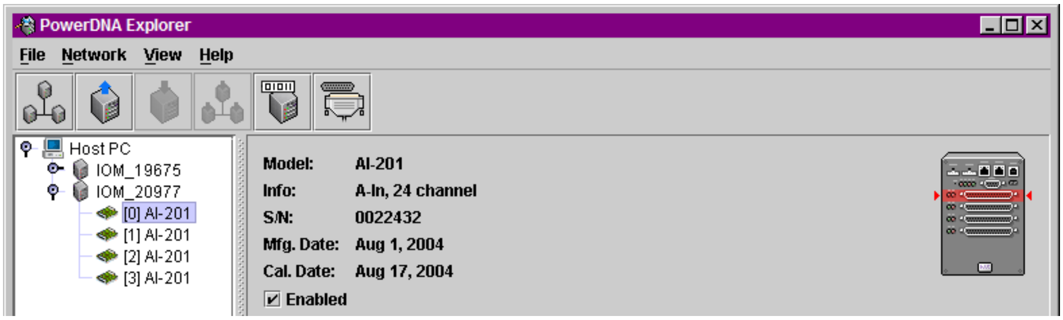
S/N shows the IOM's serial number.

MAC shows the MAC address. It cannot be changed, and thus is informational only.

IP Address shows the IOM's IP address. It can be changed.

Mode shows the mode the PowerDNA cube is in: Initialization, Configuration, Operation, or Shutdown. These modes are described in the section *IOM Modes*.

3.5.2 Device Layer Settings



Model: AI-201
Info: A-In, 24 channel
S/N: 0022432
Mfg. Date: Aug 1, 2004
Cal. Date: Aug 17, 2004
[✓] Enabled

Example Device Layer Settings for a layer

Each layer has the following settings.

Model shows the layer's model number.

Info shows some key features of the layer: *A* for analog, *D* for digital, *In* for input, *Out* for output, and a number of channels available.

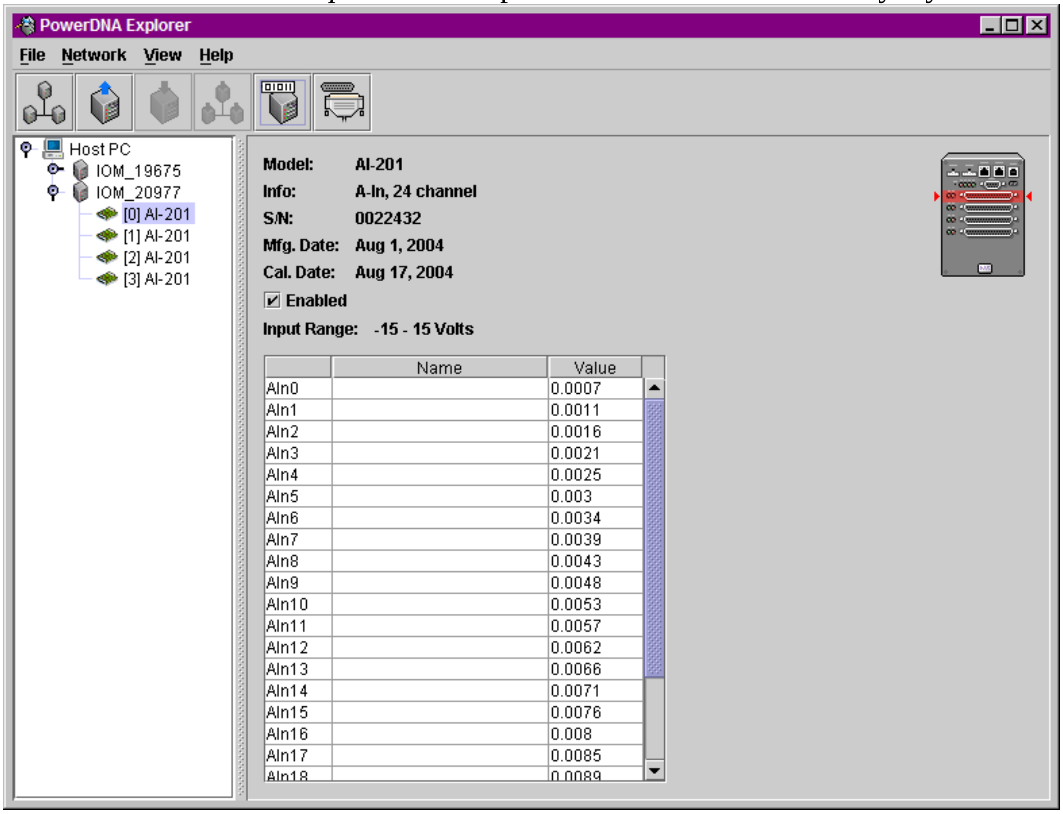
S/N shows the layer's serial number.

Mfg. Date shows the manufacturing date of the layer.

Cal. Date shows the date of the last calibration done to the layer.

Enabled is a checkbox which, when unchecked, excludes the device from configuration. The device is excluded from the Store All Configs command, and the Reload Config command is disabled. Also, the device appears grey in the tree. All devices are enabled by default.

Select *Network→Read Input Data* to update the Value column of any layer.



Model: AI-201
Info: A-In, 24 channel
S/N: 0022432
Mfg. Date: Aug 1, 2004
Cal. Date: Aug 17, 2004
☑ Enabled
Input Range: -15 - 15 Volts

	Name	Value
AIn0		0.0007
AIn1		0.0011
AIn2		0.0016
AIn3		0.0021
AIn4		0.0025
AIn5		0.003
AIn6		0.0034
AIn7		0.0039
AIn8		0.0043
AIn9		0.0048
AIn10		0.0053
AIn11		0.0057
AIn12		0.0062
AIn13		0.0066
AIn14		0.0071
AIn15		0.0076
AIn16		0.008
AIn17		0.0085
AIn18		0.0089

Example after network→Read Input Data

At this screen you can edit channel names. After editing names, choose Network→Store Config to save changes to the layer. This is true for all layers.

Also, if the user has changed a configuration value but has not chosen Network→Store Config to save them, previous values can be re-read from the layer using Network→Reload Config.

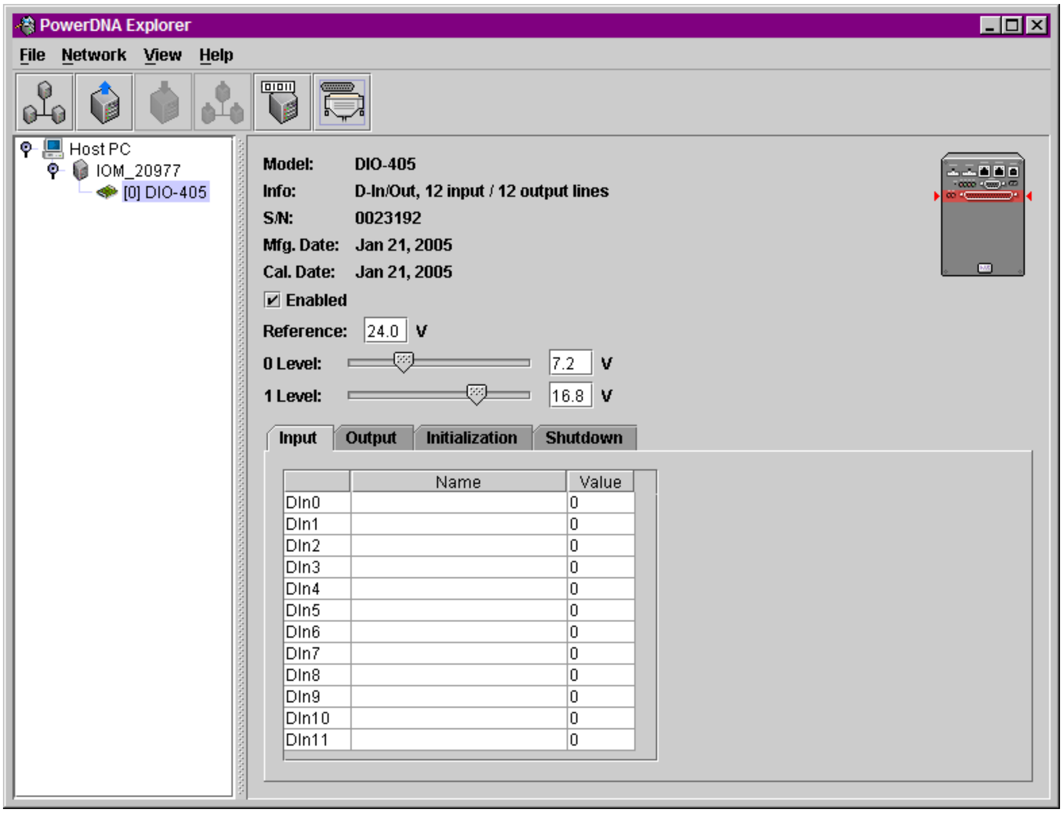
AI-205 and AI-225 layer screens are same as the AI-201 layer, just with different input range and number of channels.

In addition, digital and analog output layers have settings specific to their layer type.

1 Digital Input/Output Layer Settings

We'll use the DIO-405 as an example to start with, then show how the DI-401, DO-402 and DIO-403 are different.

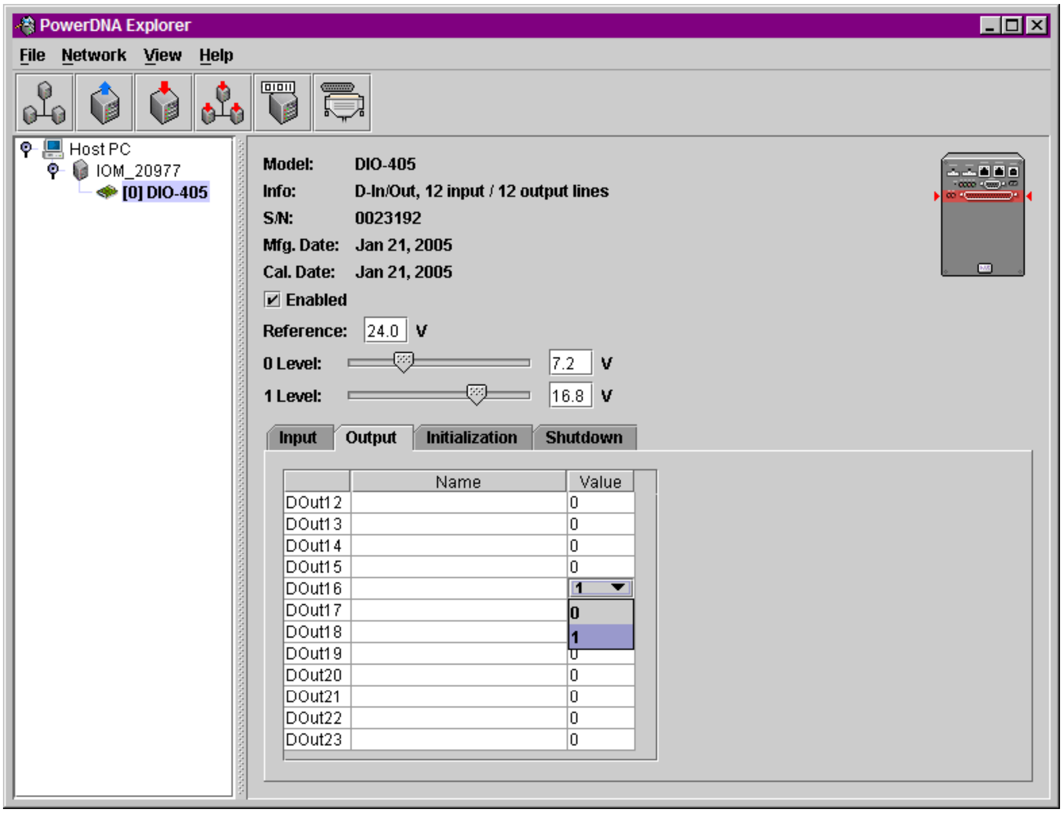
Note: Use *Network→Read Input Data* to see immediate input values in Input tabs. Use *Network→Store Config* to save values to the layer.



Model: DIO-405
Info: D-In/Out, 12 input / 12 output lines
S/N: 0023192
Mfg. Date: Jan 21, 2005
Cal. Date: Jan 21, 2005
☑ Enabled
Reference: 24.0 V
0 Level: 7.2 V
1 Level: 16.8 V

	Name	Value
DIn0		0
DIn1		0
DIn2		0
DIn3		0
DIn4		0
DIn5		0
DIn6		0
DIn7		0
DIn8		0
DIn9		0
DIn10		0
DIn11		0

Example DIO-405 layer inputs



Model: DIO-405
Info: D-In/Out, 12 input / 12 output lines
S/N: 0023192
Mfg. Date: Jan 21, 2005
Cal. Date: Jan 21, 2005
☑ Enabled
Reference: 24.0 V
0 Level: 7.2 V
1 Level: 16.8 V

	Name	Value
DOut12		0
DOut13		0
DOut14		0
DOut15		0
DOut16		1
DOut17		0
DOut18		1
DOut19		0
DOut20		0
DOut21		0
DOut22		0
DOut23		0

Example DIO-405 layer outputs

Reference is a reference voltage.

0 level/1 level are hysteresis values described fully in the DIO-401/2/5 section of this manual.

Input/Output/Initialization/Shutdown tabs switch between settings for init and shutdown states, as well as operation mode configuration, and display of current data.

All tabs contain the following columns:

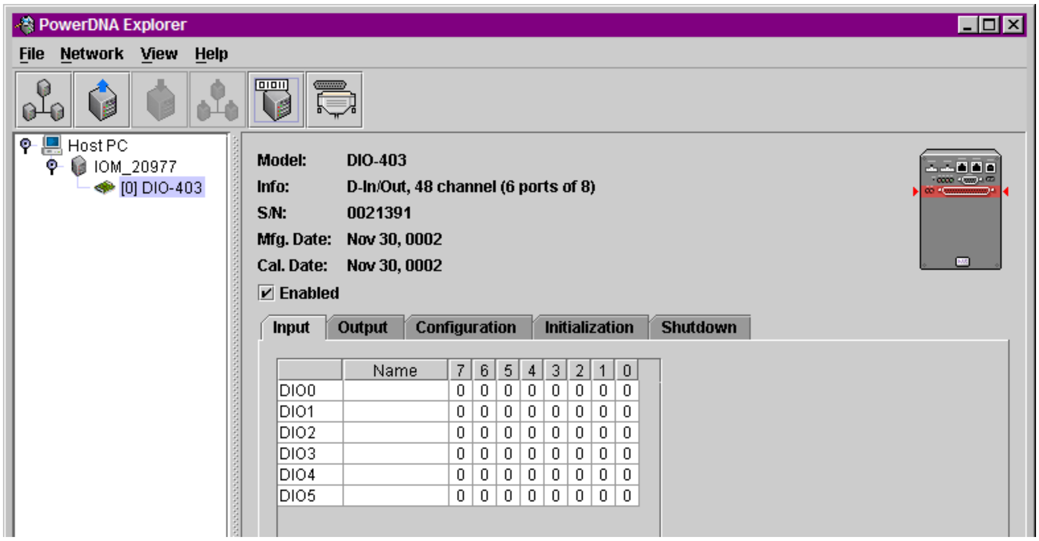
- The unnamed first column contains the channels.
- **Name** is a user-defined string.

- **Value** contains 0 or 1. It is a drop-down menu for output channels allowing the user to select 0 or 1.

The DI-401 layer just has Reference and 0 and 1 Level controls, and Input tab.

The DO-402 layer just has Output, Initialization, and Shutdown tabs; no Reference value or Level sliders.

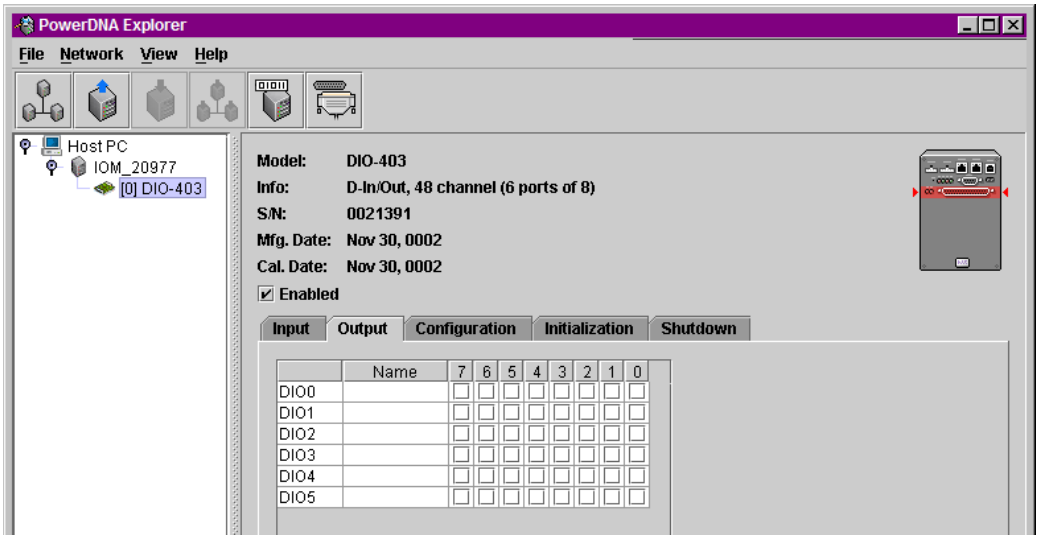
The DIO-403 layer is different because it groups 8-bits at a time into ports, and three ports into two channels. For the sake of abstraction in PowerDNA Explorer, we'll call all the ports channels.



Model: DIO-403
Info: D-In/Out, 48 channel (6 ports of 8)
S/N: 0021391
Mfg. Date: Nov 30, 0002
Cal. Date: Nov 30, 0002
☑ Enabled

	Name	7	6	5	4	3	2	1	0
DIO0		0	0	0	0	0	0	0	0
DIO1		0	0	0	0	0	0	0	0
DIO2		0	0	0	0	0	0	0	0
DIO3		0	0	0	0	0	0	0	0
DIO4		0	0	0	0	0	0	0	0
DIO5		0	0	0	0	0	0	0	0

Example DIO-403 layer inputs



Model: DIO-403
Info: D-In/Out, 48 channel (6 ports of 8)
S/N: 0021391
Mfg. Date: Nov 30, 0002
Cal. Date: Nov 30, 0002
☑ Enabled

Input | Output | Configuration | Initialization | Shutdown

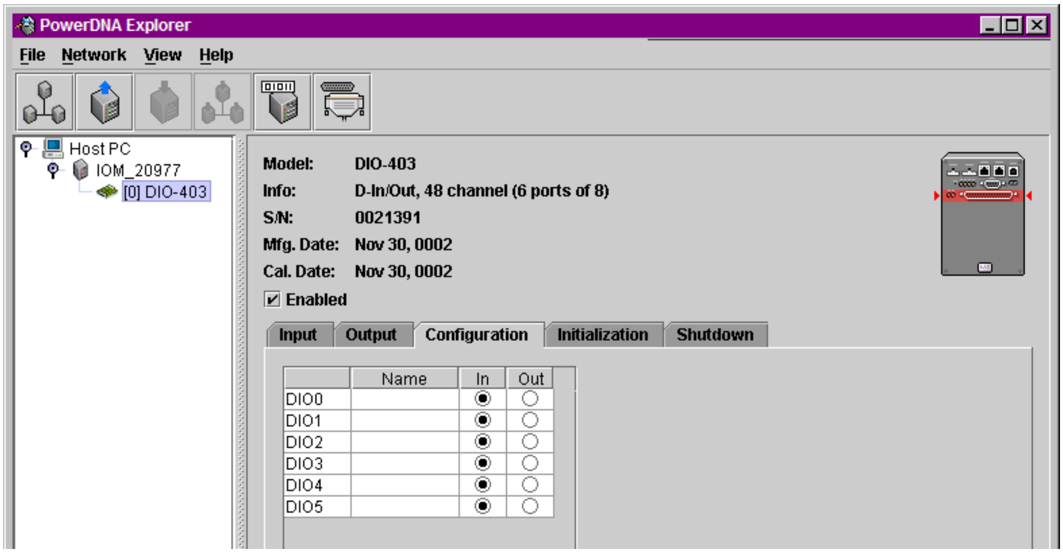
	Name	7	6	5	4	3	2	1	0
DIO0		☐	☐	☐	☐	☐	☐	☐	☐
DIO1		☐	☐	☐	☐	☐	☐	☐	☐
DIO2		☐	☐	☐	☐	☐	☐	☐	☐
DIO3		☐	☐	☐	☐	☐	☐	☐	☐
DIO4		☐	☐	☐	☐	☐	☐	☐	☐
DIO5		☐	☐	☐	☐	☐	☐	☐	☐

Example DIO-403 layer outputs

Input/Output/Configuration/Initialization/Shutdown tabs switch between settings for init and shutdown states, as well as operation mode configuration, and display of current data.

Input/Output tabs get/set the current input/output values. They contain the following columns:

- The unnamed first column contains the channels.
- **Name** is a user-defined string.
- **7 through 0** contain the values 0 or 1. For the output tab, they are checkmarks for output channels allowing the user to select 0 (unchecked) or 1 (checked).



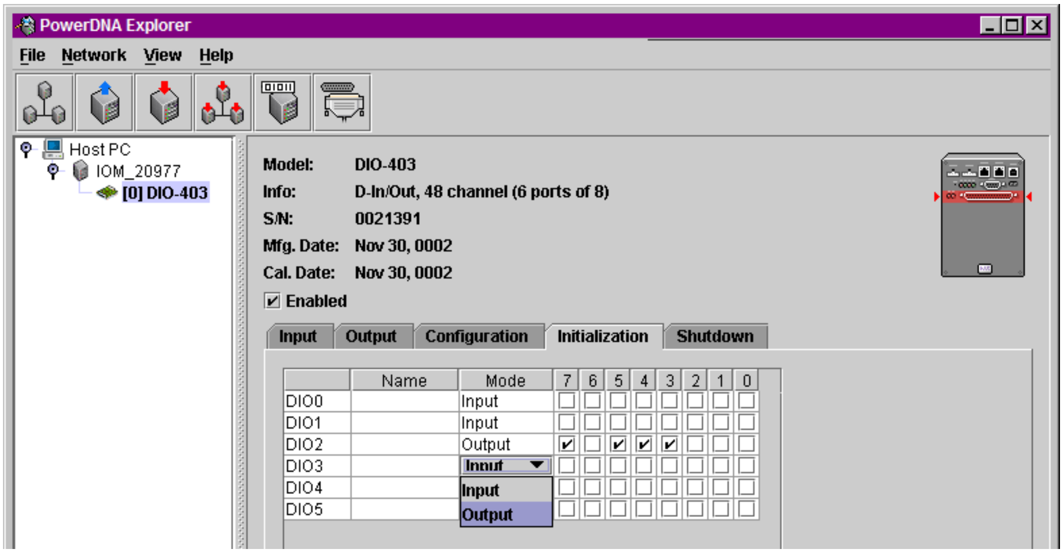
Model: DIO-403
Info: D-In/Out, 48 channel (6 ports of 8)
S/N: 0021391
Mfg. Date: Nov 30, 0002
Cal. Date: Nov 30, 0002
☑ Enabled

	Name	In	Out
DIO0		●	○
DIO1		●	○
DIO2		●	○
DIO3		●	○
DIO4		●	○
DIO5		●	○

Example DIO-403 layer configuration

Configuration tab gets/sets the current input/output directions per port. It contain the following columns:

- The unnamed first column contains the channels.
- **Name** is a user-defined string.
- **In/Out** contain toggle switches to select whether the channel is to be used for input or for output.



Model: DIO-403
Info: D-In/Out, 48 channel (6 ports of 8)
S/N: 0021391
Mfg. Date: Nov 30, 0002
Cal. Date: Nov 30, 0002
☑ Enabled

Input | Output | Configuration | Initialization | Shutdown

	Name	Mode	7	6	5	4	3	2	1	0
DIO0		Input	☐	☐	☐	☐	☐	☐	☐	☐
DIO1		Input	☐	☐	☐	☐	☐	☐	☐	☐
DIO2		Output	✓	☐	✓	✓	✓	☐	☐	☐
DIO3		Input	☐	☐	☐	☐	☐	☐	☐	☐
DIO4		Input	☐	☐	☐	☐	☐	☐	☐	☐
DIO5		Output	☐	☐	☐	☐	☐	☐	☐	☐

Example DIO-403 layer initialization

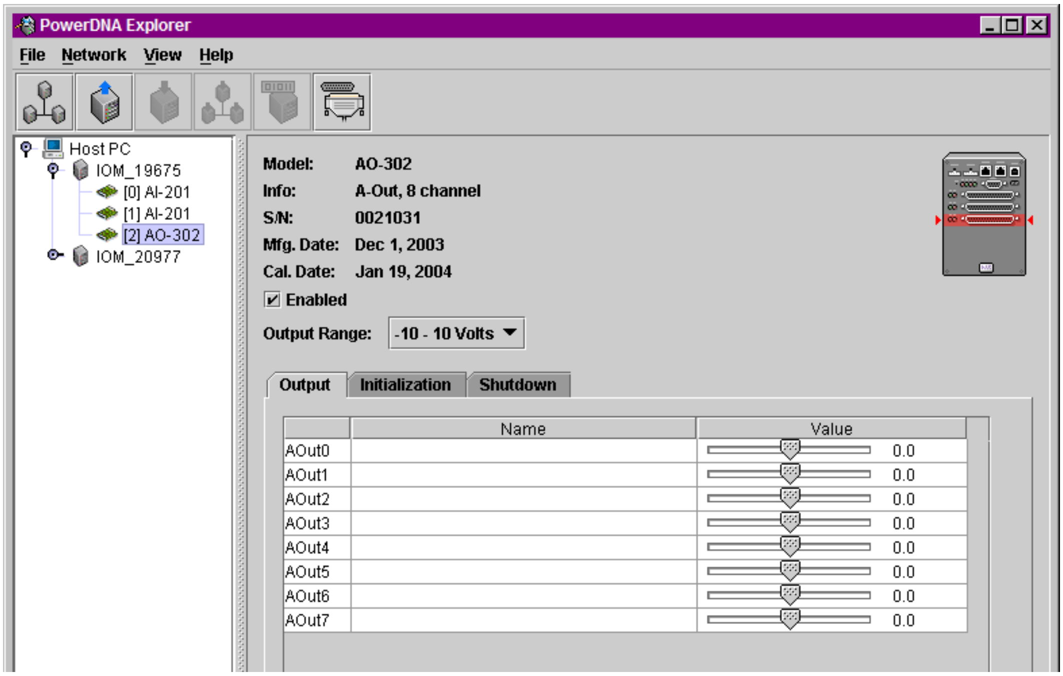
Initialization/Shutdown tabs allow the user to set port as input or output, and set output values. They contain the following columns:

- The unnamed first column contains the channels.
- **Name** is a user-defined string.
- **Mode** specifies whether the channel is input or output.
- **7 through 0** contain the values 0 or 1. They are checkmarks for output channels allowing the user to select 0 (unchecked) or 1 (checked).

2 Analog Output Layer Settings

We'll use the AO-302 as an example.

Note: Use *Network→Read Input Data* to see immediate input values in Input tabs. Use *Network→Store Config* to save values to the layer.



Model: AO-302
Info: A-Out, 8 channel
S/N: 0021031
Mfg. Date: Dec 1, 2003
Cal. Date: Jan 19, 2004
☑ Enabled
Output Range: -10 - 10 Volts

	Name	Value
AOut0		0.0
AOut1		0.0
AOut2		0.0
AOut3		0.0
AOut4		0.0
AOut5		0.0
AOut6		0.0
AOut7		0.0

Example AO-302 layer

The user can change output, initialization, and shutdown values. User can also change Output Range using the combo box, and this only affects values displayed in initialization and shutdown tabs. The user can then choose *Network→Store Config* to apply all changes to the layer.

Output Range is a popup allowing the user to choose between -10...0V, 0...+10V, and -10...+10V.

Output/Initialization/Shutdown tabs switch between settings for init and shutdown states, as well as operation mode configuration.

The **Output**, **Initialization** and **Shutdown** tabs contain the channel list table, which has the following columns:

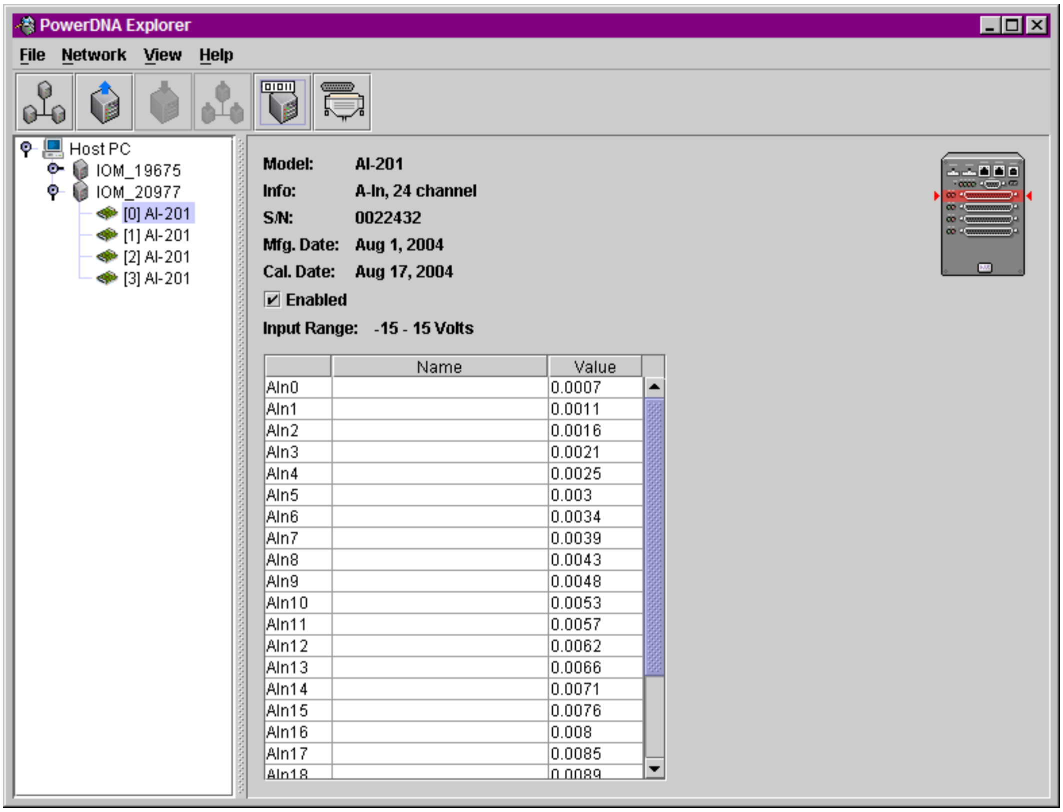
- The unnamed first column contains the channels.
- **Name** is a user-defined string.

- **Value** contains a slider to set the voltage to output from the channel and the numerical voltage value, which the user can input directly. The actual voltage depends on the selected output range.

3 Analog Input Layer Settings

We'll use the AI-201 as an example to start with. The AI-202 and AI-205 are similar.

Note: Use *Network→Read Input Data* to see immediate input values in Input tabs. Use *Network→Store Config* to save values to the layer.



Model: AI-201
Info: A-In, 24 channel
S/N: 0022432
Mfg. Date: Aug 1, 2004
Cal. Date: Aug 17, 2004
☑ Enabled
Input Range: -15 - 15 Volts

	Name	Value
AIn0		0.0007
AIn1		0.0011
AIn2		0.0016
AIn3		0.0021
AIn4		0.0025
AIn5		0.003
AIn6		0.0034
AIn7		0.0039
AIn8		0.0043
AIn9		0.0048
AIn10		0.0053
AIn11		0.0057
AIn12		0.0062
AIn13		0.0066
AIn14		0.0071
AIn15		0.0076
AIn16		0.008
AIn17		0.0085
AIn18		0.0089

Example AI-201 layer

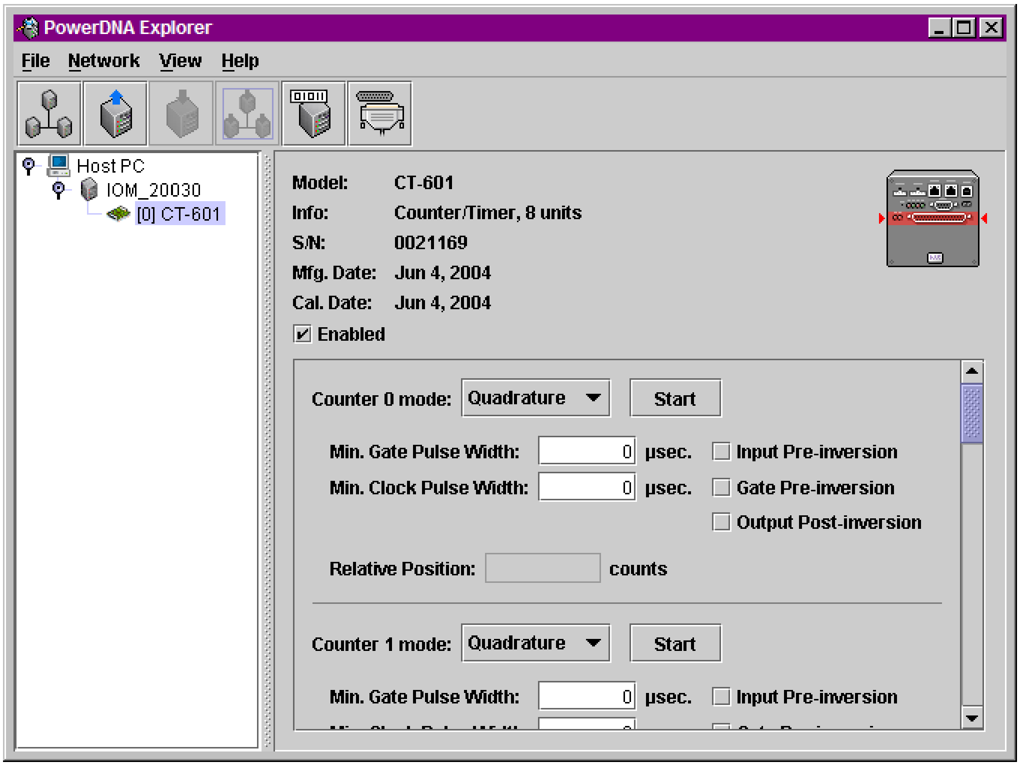
Input Range shows the specified input range. It cannot be changed, and thus is informational only.

The Data table contains the values currently coming into the device. The table is initially blank until the user hits invokes Refresh Data, unless auto-refresh is activated in the preferences dialog. The table has three columns:

- The unnamed first column contains the channels.
- **Name** is a user-defined string.
- **Value** shows the current value.

4 Counter/Timer Layer Settings

We'll use the CT-601 as an example.



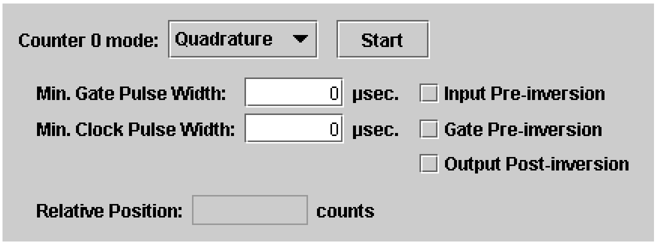
Model: CT-601
Info: Counter/Timer, 8 units
S/N: 0021169
Mfg. Date: Jun 4, 2004
Cal. Date: Jun 4, 2004
☑ Enabled

Counter 0 mode: Quadrature
Min. Gate Pulse Width: 0 µsec. ☐ Input Pre-inversion
Min. Clock Pulse Width: 0 µsec. ☐ Gate Pre-inversion
☐ Output Post-inversion
Relative Position: counts

Counter 1 mode: Quadrature
Min. Gate Pulse Width: 0 µsec. ☐ Input Pre-inversion

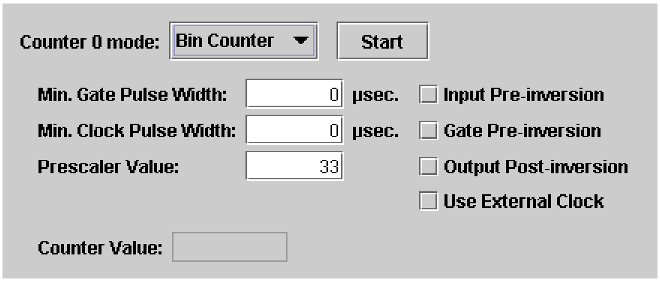
Example CT-601 layer

The CT-601 layer has 8 counters. Each counter can be set to one of four different modes: Quadrature, Bin Counter, Pulse Width Modulation (PWM), or Pulse Period. When you change the mode of a counter using the mode combo box, the controls for that counter will change to those appropriate for the mode.


Counter 0 mode: Quadrature
Start

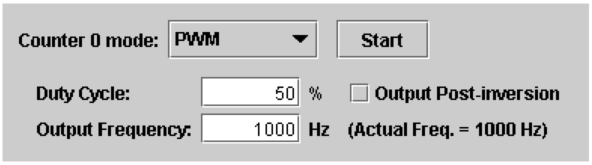
Min. Gate Pulse Width:	0	µsec.	☐ Input Pre-inversion
Min. Clock Pulse Width:	0	µsec.	☐ Gate Pre-inversion
			☐ Output Post-inversion
Relative Position:		counts	

Example Quadrature controls


Counter 0 mode: Bin Counter
Start

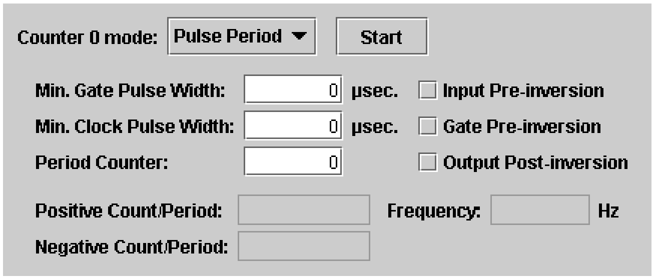
Min. Gate Pulse Width:	0	µsec.	☐ Input Pre-inversion
Min. Clock Pulse Width:	0	µsec.	☐ Gate Pre-inversion
Prescaler Value:	33		☐ Output Post-inversion
			☐ Use External Clock
Counter Value:			

Example Bin Counter controls


Counter 0 mode: PWM
Start

Duty Cycle:	50	%	☐ Output Post-inversion
Output Frequency:	1000	Hz	(Actual Freq. = 1000 Hz)

Example Pulse Width Modulation (PWM) controls

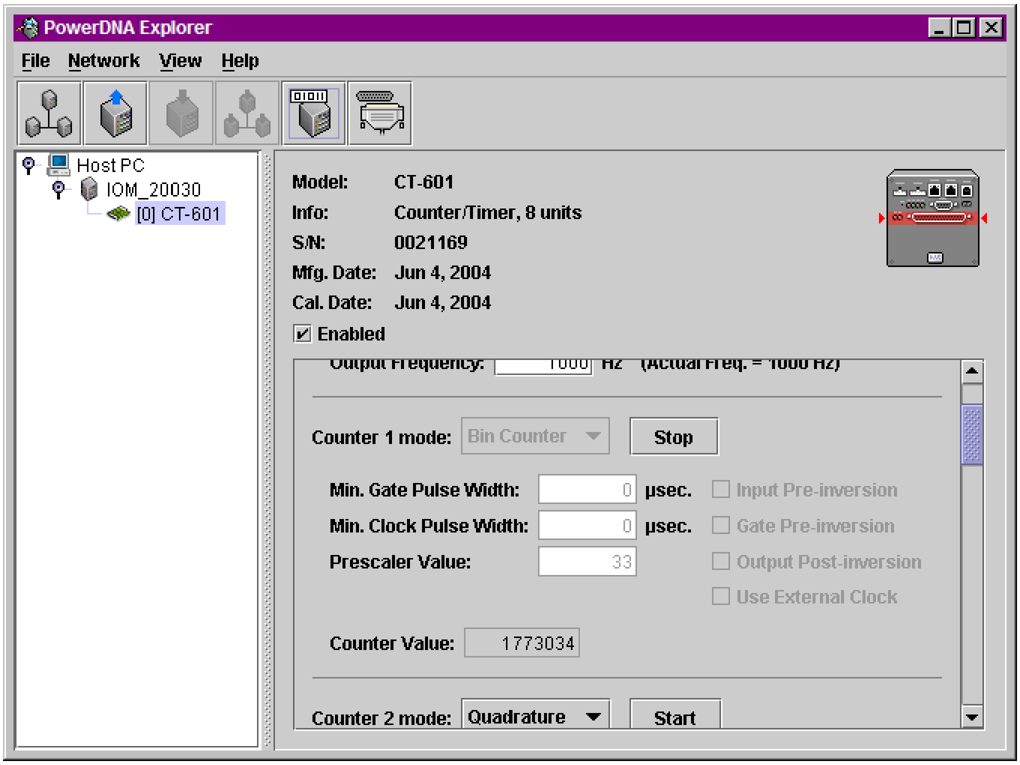

Counter 0 mode: Pulse Period
Start

Min. Gate Pulse Width:	0	µsec.	☐ Input Pre-inversion
Min. Clock Pulse Width:	0	µsec.	☐ Gate Pre-inversion
Period Counter:	0		☐ Output Post-inversion
Positive Count/Period:		Frequency:	Hz
Negative Count/Period:			

Example Pulse Period controls

After setting the configuration for a counter, you can choose *Network→Store Config* to store the settings on the device. Clicking the *Start* button will also write the user configuration to the layer.

Clicking the *Start* button for a counter will start that counter on the layer. The Start button will turn into a Stop button, and the other controls for that counter will become disabled until you click Stop. While the layer is running, you can choose *Network→Read Input Data* to retrieve runtime values from the counter, which will display in the read-only text field(s) of the counter control panel.



Model: CT-601
Info: Counter/Timer, 8 units
S/N: 0021169
Mfg. Date: Jun 4, 2004
Cal. Date: Jun 4, 2004
☑ Enabled

Output Frequency: 1000 Hz (Actual Freq. = 1000 Hz)

Counter 1 mode: Bin Counter | Stop

Min. Gate Pulse Width: 0 µsec. ☐ Input Pre-inversion
Min. Clock Pulse Width: 0 µsec. ☐ Gate Pre-inversion
Prescaler Value: 33 ☐ Output Post-inversion
☐ Use External Clock

Counter Value: 1773034

Counter 2 mode: Quadrature | Start

Example of started counter

4 The PowerDNA Core Module

The top two slots of any 5- or 8-slot cube are occupied by the Core Module. The Core Module consists of a CPU and peripheral devices (RS-232, NIC, SD, etc).

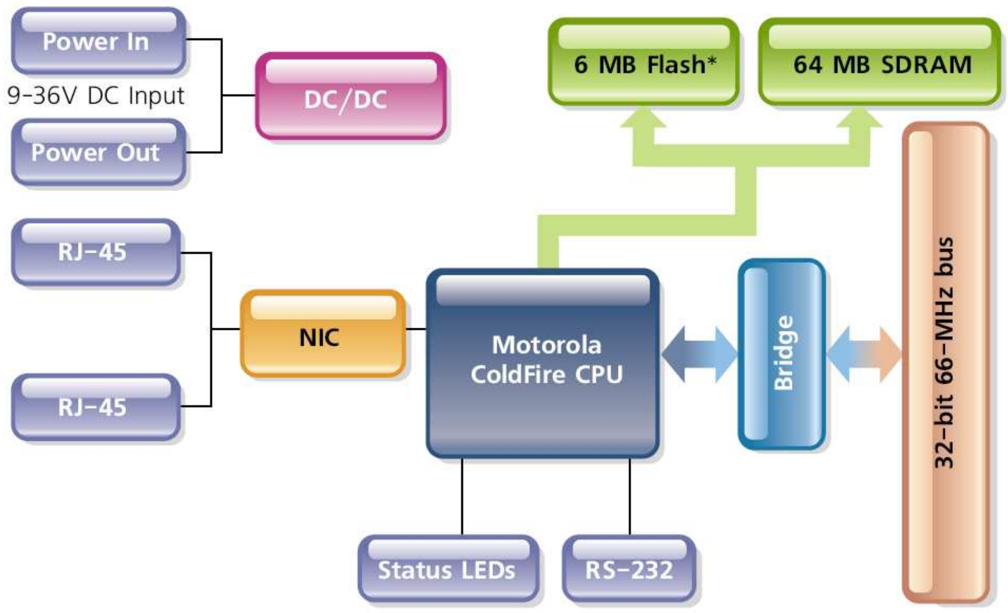
The NIC is either a copper (100BaseT), or a Fiber-optic (10/100Base-FX) interface. The CPU is either FreeScale ColdFire (DNA-CM) or PowerPC CPU (DNA-PPC).

In addition, an RS-232 port is provided for configuration, and activity lights for status.

This chapter focuses on the device architecture of the Core Module - no layers.

4.1.1 Device architecture of DNA-CM

CM controller architecture can be represented as follows:



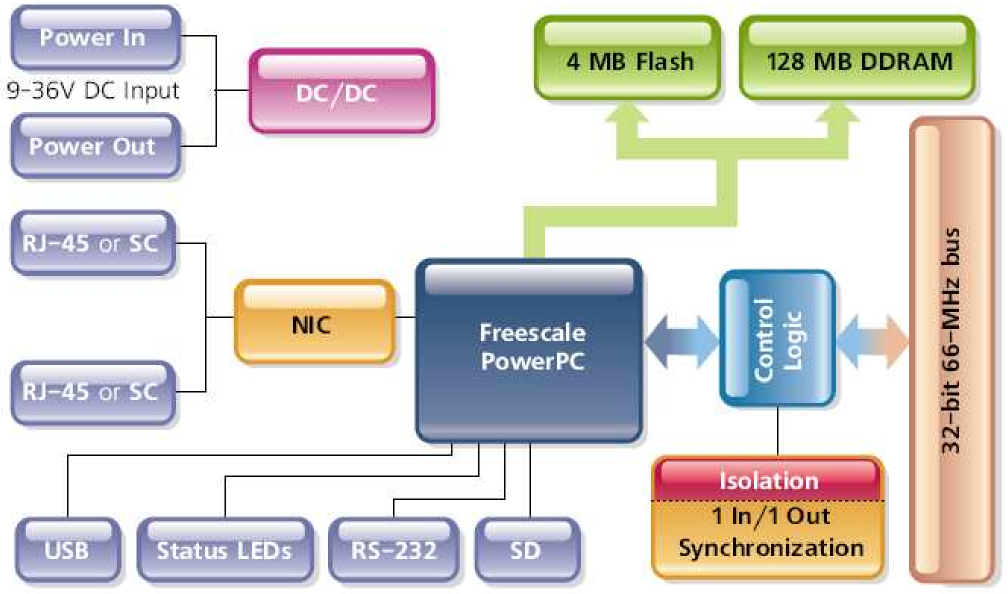
The core of the system is a FreeScale (formerly Motorola) ColdFire MCF5272 processor. The processor is directly connected to the following components:

- Network interface MII port
- RS-232 port
- IrDA port
- 2MB user flash memory
- 4MB system flash memory
- 64MB of SDRAM
- Bus bridge
- Control logic
- LEDs
- Watchdog timer with real-time clock (battery backed)

Not all components are available for control from the CPU. The CPU can program flash memory, set the LEDs, set up the watchdog timer, set the real-time clock and use 256 bytes of backed-up memory in the watchdog timer chip.
All functions are available at the firmware level only (described in iom.c/iom.h).

4.1.2 Device architecture of DNA-PPC

PowerPC controller architecture can be represented as follows:



The core of the system is a FreeScale PowerPC MPC5200 400MHz processor. The processor controls the following components:

- Network interface MII port
- RS-232 port
- SYNC port
- 4MB system flash memory
- 128MB of 266MHz DDRAM
- Bus bridge

All functions are available at the firmware level only (iom.c/iom.h).

5 Programming layer specific functions

5.1.1 Memory map

ColdFire-based CM has the following memory map:

Device	Start address	End address	Size	Description
SDRAM	0x0	0x4000000	64MB	SDRAM_ADDRESS
Interrupt table	0x0	0x400	1024	Processor address map
Firmware load address	0x20000			End address and size depends on firmware size
Firmware start address	0x20400			First execution instruction of firmware
IMM	0x10000000			Memory map register - IMM_ADDRESS
On-board PLD	0x60000000	0x61000000	1MB	EXT_SRAM_ADDRESS
Watchdog timer	0x60008000			IOM_WDTIMER – within PLD access space
Processor RAMBAR	0x80000000			
User flash memory	0x90000000	0x90400000	4MB	FLASHAUX_ADDRESS
Layer – CS2	0xA0000000	0xA00FFFFC	1MB	EXT_DEV_ADDRESS2
Layer – CS3	0xA0100000	0xAFFFFFFC	256M	EXT_DEV_ADDRESS3

PowerPC-based CM has the following memory map:

Device	Start address	End address	Size	Description
SDRAM	0x0	0x8000000	128MB	SDRAM_ADDRESS
Exception table	0x0	0x3000	12k	Processor address map
IMM	0x10000000			Memory map

				register - `IMM_ADDRESS`
On-board logic	0xA00E0000	0xA00EFFFC	64kB	`EXT_SRAM_ADDRESS`
Watchdog timer	0xA00E8000			`IOM_WDTIMER` – within PLD access space
Processor RAMBAR	0x80000000			
Layer – CS2	0xA0000000	0xA00FFFFC	1MB	`EXT_DEV_ADDRESS2`
Layer – CS3	0xA0100000	0xAFFFFFFC	256M	`EXT_DEV_ADDRESS3`
Flash (parameters)	0xFFC00000	0xFFC0FFFF	64kB	`Parameters (64 sectors)`
Flash (firmware)	0xFFC10000	0xFFEFFFFF	3MB	`Firmare (3MB – 64kB)`
Falsh (U-Boot)	0xFFF00000	0xFFFFFFFF	1MB	`U-Boot`

Two address ranges are interesting for host software:
Layer address space (0xA0000000 – 0xA00FFFFC and 0xA0100000 – 0xAFFFFFFC). First address range is dedicated for devices located on CS2 line and accommodates sixteen layers with 64k memory map each. Second address range is designated for fast devices located in CS3 line and accommodates fifteen devices with 16MB memory map each.

5.1.2 Startup sequence (DNA-CM-5/8)

After reset, the processor starts monitor execution from flash memory. The monitor initializes the processor and the address map, retrieves information from the parameter sector of the flash memory and tests system memory and other system resources.
If "fwgo" parameter is set to "autorun," then the monitor waits for three seconds for the user to send Ctrl-A (which is transmitted over the serial interface.) If sent, the monitor aborts loading firmware into memory and brings up the monitor command prompt (to load a new firmware, for example).
Otherwise, the monitor reads the firmware from the flash memory and stores it in RAM. Then, the monitor executes the firmware.
Following parameters are critical to get firmware copied and started from the proper address:

```
    fwad: 0xFFE40000
    fwgo: 0x1
    fwsz: 0x100000
    fwcp: 0x20000
    fwst: 0x20400
```

These parameters can be reviewed using “show” command while at the monitor “#>” prompt.
“fwad” is the initial address where firmware is stored. This address shall be set before storing firmware or executing it.
“fwgo” defines whether the monitor should load firmware (1) or should display a command prompt.
“fwsz” defines the size of the stored firmware. Default value is 0x100000 – one megabyte.
“fwcp” defines the address to which the monitor copies firmware from flash memory. The default is 0x20000. Firmware is compiled to run from this address.
“fwst” defines firmware entry point. Firmware entry point follows vector table and is located with offset 0x400 from the beginning of the firmware code.
These parameters are pre-programmed at the factory and there is no known reason for user to change them.
The monitor command “fwjmp” causes monitor to load and execute firmware.

5.1.3 Startup sequence (DNA-PPC-5/8)

After reset processor reads boot-up sequence located at 0xfffff100. This command sequence is a part of U-Boot code. U-Boot initializes all major subsystems of the CM including DDRAM and Ethernet interface.
After initializing U-Boot performs command list stored in its environment sector under the **bootcmd** entry. Standard commands to launch firmware are etiher **fwjmp** or **go 0xffc10000** depends on version of U-Boot installed. Then, U-Boot gives up control to the firmware code located at 0xffc10000. Firmware self-expands into the DDRAM, initializes exception table and starts execution.

5.1.4 Interfacing to the CM module using a serial interface

There are two ways to set up CM parameters. The first one is the use of serial interface and the second one is the use of DaqBIOS calls.
To connect to the serial interface, the user should connect **extender** 9-wires serial cable to the PowerDNA cube (plug male connector) and your PC COM1 serial port (plug female connector). Some cables have female-to-female connector so the user should use a gender-changer.
Set up your terminal to the proper serial port, 57600 bit rate, no parity, eight data bits and one stop bit.

Alternately, using *Start→Run…*on the Microsoft Windows desktop, type *\Program Files\UEI\PowerDNA\Firmware\mttty.exe*. Click *File→Connect*. Once a connection to the PowerDNA cube is established, tap “Enter” once. The PowerDNA cube should respond with either a “`DQ>`” prompt (this is firmware prompt) or a “`#>`” prompt (monitor prompt).
Once you see the “`DQ>`” prompt, you can type “help<enter>” to receive the list of all available commands.

Following commands are available:

```
DQ> help

    help Display this help message      help
     set Set parameter                  set option value
    show Show parameters                show
   store Store parameters (flash)       store
      mw Write wr <addr> <val> (hex)    mw
      mr Read rd <addr> (hex)           mr
    time Show/Set time                  time [mm/dd/yyyy] [hh:mm:ss]
    pswd Set password                   pswd {user|su}
      ps Show process state #           ps [value]
    test Test something                 test [test number]
   simod System Init/Module Cal         simod [routine]
   reset Reset system                   reset [all]
  dqping Send DQ_ECHO to <mac addr>     dqping [MAC|IP]
    mode Set current mode               mode {init|config|oper|shutdown}
[ID]
     log Display log content            log [start [end]] -1 = clear
     ver Show firmware version          ver
  devtbl Show all devices/layers        devtbl
 netstat Show network statistics        netstat
```

One of the most useful commands is “show”:

```
DQ> show

        name: "IOM_22811"
       model: 0x1005
      serial: 0022811
         mac: 00:0C:94:00:59:1B
        fwct: 1.2.0.0
         srv: 192.168.0.229
          ip: 192.168.0.67
     gateway: 192.168.0.1
     netmask: 255.255.255.0
         udp: 6334
```

This command display current values of every major PowerDNA cube parameter.

To change parameters use “set” command (type set for “set” command syntax).

```
DQ> set

Valid 'set' options:
        name: <Device name>
       model: <Model id>
      serial: <Serial #>
         mac: <my ethernet address>
        fwct: <autorun.runtype.portnum.umports>
         srv: <Host IP address>
          ip: <IOM IP address>
     gateway: <gateway IP address>
     netmask: <netmask IP address>
         udp: <udp port (dec)>
```

For example, to set new IP address one may type:

```
DQ> set ip 192.168.100.100
```

Other parameters can be changed the same way. Once parameters are set, you have to store them into non-volatile flash memory:

```
DQ> store
Flash: 1212 bytes of 1212 stored! CRC=0x8975E34A Old=0x8975E34A
Configuration stored
DQ>
```

After parameters are stored, the user should reset firmware (start firmware execution from the beginning without full hardware reset):

```
DQ> reset
Stopping…

DaqBIOS (C) UEI, 2001-2004. Running PowerDNA Firmware
Built on 16:39:15 Oct  1 2004
Initialize uC/OS-II (Real-Time Kernel v.252)
Configuration recalled
3 device detected

Address    Irq  Model Option  Phy/Virt  S/N    Pri  DevN
---------------------------------------------------------
0xA0000000   2    205    1    phys   0023115      10  0
0xA0010000   2    205    1    phys   0023117      20  1
0xA0020000   2    205    1    phys   0023119      30  2
---------------------------------------------------------
Current time: 18:53:45 11/01/2004
IOM: TCP/IP/DQ stack. MAC=00:0C:94:00:59:1B
```

To perform full hardware reset use:

```
DQ> reset all
```

The full reset performs physical reset of the CPU and initiates the whole startup sequence.

Some commands (mr, mw, set and store particularly) require entering a user password. Once the password is entered these commands become enabled until firmware reset. There are two level of password protection available. First is user level and second is super-user level. Super-user level is currently used only for updating firmware over the Ethernet link.

`DQ> pswd user` sets up user level password. First, you'll be asked about old password and then (if matches) to enter new password twice.

`DQ> pswd su` sets up super-user level password. First, you'll be asked about old super-user password and then (if matches) to enter new super-user password twice.

PowerDNA cubes come with the default password set to "powerdna".

Some DaqBIOS commands require clearing up user or super-user password. Use DqCmdSetPassword() before calling these functions. The *PowerDNA API Reference Manual* notes which functions are password-protected.

Another useful command is "devtbl". This command displays all I/O layer found and initialized by firmware along with assigned device numbers.
Use these device numbers in host software to address these devices.
Priority tells in which order device drivers are located in the device stack. Device with lower priority number receives shared interrupt first. The firmware sets up device driver priorities when it registers device drivers.

"simod" is a command for system initialization and module calibration.
"simod 0" is used to initialize initial layer parameters – serial number, option, etc. We do not recommend use of this command in the field.
"simod 1" allows layer calibration. Different layers have different calibration procedures explained in respective sections of this document.
"simod 3" allows to perform factory tests – this is non-destructive command.

WARNING: Once you use the "simod 0" command layer warranty is void.

WARNING: Once you use the “simod 1” command layer calibration warranty is void.

5.1.5 Setting parameters

Using the serial interface, one can set up following parameters:

```
        name: <Device name>
       model: <Model id>
      serial: <Serial #>
         mac: <my ethernet address>
        fwct: <autorun.runtype.portnum.umports>
         srv: <Host IP address>
          ip: <IOM IP address>
     gateway: <gateway IP address>
     netmask: <network mask>
         udp: <udp port>
```

“Name” sets the device name (up to 32 characters)

“Model” sets the device model (factory programmed, do not change). Valid values are 0x1005 – 100-Base-T five layer PowerDNA cube, 0x1008 – 100-Base-T eight layer PowerDNA cube, 0x1105 – 100-Base-FX (fiber optics) five layer PowerDNA cube, 0x1108 – 100-Base-FX eight layer PowerDNA cube.

“Serial” sets the PowerDNA cube’s serial number (factory programmed, do not change)

“MAC” sets the PowerDNA cube’s MAC Ethernet address (factory programmed, do not change)

“fwct” defines the behavior of the monitor upon boot-up. Valid values for “autorun” are zero – stay in monitor after initial boot sequence, and one – copy firmware to SDRAM memory location and execute from there. Valid values for “runtype” are TYPE_IOM=2,
TYPE_AUTO =4 and TYPE_SA=8. For normal operation, use TYPE_IOM. “portnum” and “umports” parameters are reserved in the current release and should be set to zero.

“Srv” sets the host IP address. You have to set the host IP address only if raw Ethernet protocol is in use (used in homogenous IOM networks only.) This parameter is ignored when the PowerDNA cube is used over the UDP protocol or from the host.

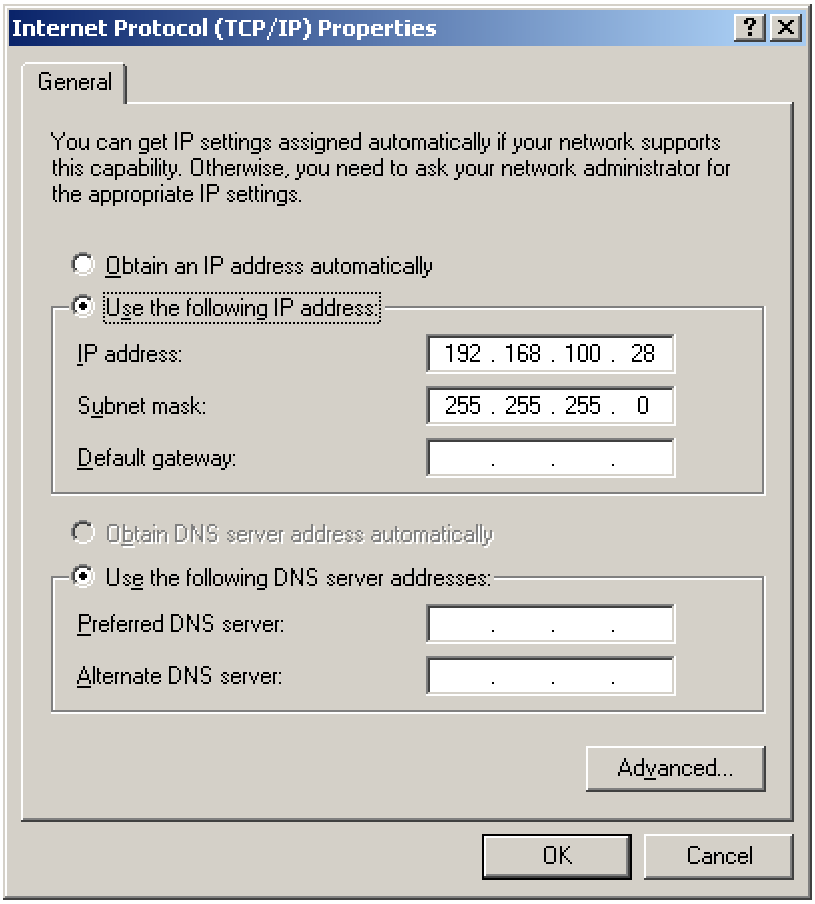
“IP” specifies the IOM IP address. This is the most important parameter the user should change to allow the PowerDNA cube to be visible on the network. The PowerDNA cube responses to every UDP packet contains a DaqBIOS prolog sent to this address. The current release does not support DHCP, thus the user should set up the IP address.

“gateway” specifies where the PowerDNA cube should send an IP packet if a requested IP packet exist outside of the PowerDNA cube network (defined by the network mask). Ask system administrator if you use your PowerDNA cube on the office network.

“netmask” specifies what type of subnet the PowerDNA cube is connected to. Factory sets netmask to type C IP network – 254 nodes maximum

“udp” specifies what port the firmware should use if a network packet originated from this PowerDNA cube without previous request from the host side. If the PowerDNA cube replies to a DaqBIOS packet, it uses the source IP address from the IP packet header and source UDP port from UDP packet header.

Let’s assume that user wants to connect a PowerDNA cube to the dedicated network (secondary NIC adapter in the host PC).
Let’s also assume that host IP address on this dedicated network is:
IP address: 192.168.100.28
Network mask: 255.255.255.0
Gateway: ignored
DNS: ignored



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 192 . 168 . 100 . 28
Subnet mask: 255 . 255 . 255 . 0
Default gateway: . . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server: . . .
Alternate DNS server: . . .

Set PowerDNA cube address to any address in the range of 192.168.100.1 thru 192.168.0.254 excluding 192.168.100.28 – the host IP address.

For example type:

```
DQ> set ip 192.168.0.2
```

Then:

```
DQ> store
```

This sequence of commands stores a new IP address in the flash parameter sector. Then, you have to reset the PowerDNA cube.

PowerDNA cubes come from the factory with IP addresses already preset for 192.168.x.x network. The factory IP address can be found on the label located on the back of the PowerDNA cube along with factory MAC address.

After the IP address is set, the user can establish communication with the PowerDNA cube using the PowerDNA Explorer.

5.1.6 How to update firmware

See the appendix for this information.

5.1.7 Clock and watchdog access

To show and set up the date and time use "time" command:

```
DQ> time
Current time: 17:39:22 11/01/2004
```

To set up time of the day:

```
DQ> time 17:40:00
```

To set up date:

```
DQ> time 11/03/2004
```

Date and time are stored in the battery-backed real-time clock chip.

5.2 Common Layer Interface

The Common Layer Interface is the protocol used in a PowerDNA cube for communication between the IOM and its layers.

5.2.1 Channel list

A channel list specifies what channels and in which sequence should be acquired/output. Every layer has it's own specific set of channel list flags. The firmware takes care of this hardware dependency. Please see the specific layer description to find out what channel list flags are supported.
Users should use the following flags, generalized for all layers.

```
// Channel list entries definition - lower 16 bits are reserved for channel number
// gain and special, module-specific settings
#define DQ_LNCL_NEXT    (1UL<<31)    // channel list has next entry
#define DQ_LNCL_INOUT   (1UL<<30)    // input or output subsystem
#define DQ_LNCL_SS1     (1UL<<29)    // subsystem (high)
#define DQ_LNCL_SS0     (1UL<<28)    // subsystem (low)
#define DQ_LNCL_IRQ     (1UL<<27)    // fire IRQ
```

```
#define DQ_LNCL_NOWAIT  (1UL<<26)    // execute this step but don't wait
                                     // for the next CV
#define DQ_LNCL_SKIP    (1UL<<25)    // execute this step and discard data
                                     // for the next CV
#define DQ_LNCL_CLK     (1UL<<24)    // wait for the next channel list clock
#define DQ_LNCL_CTR     (1UL<<23)    // clock counter once
#define DQ_LNCL_WRITE   (1UL<<22)    // write to the channel but not update
#define DQ_LNCL_UPDALL  (1UL<<21)    // update all written channels
#define DQ_LNCL_TSRQ    (1UL<<20)    // copy TS along with data ( i+=2 )
#define DQ_LNCL_SLOW    (1UL<<19)    // slow down operation
#define DQ_LNCL_RSVD2   (1UL<<18)    // reserved
#define DQ_LNCL_RSVD1   (1UL<<17)    // reserved
#define DQ_LNCL_RSVD0   (1UL<<16)    // reserved
#define DQ_LNCL_DIFF    (1UL<<15)    // differential mode
```

There are a few helper macros defined to simplify setting gain and subsystem flags.

```
#define DQ_LNCL_GAIN(G) ((G & 0xf)<<8)   // set gain

#define DQ_LNCL_GETGAIN(E)  ((E & 0xf00)>>8)  // pull out gain
#define DQ_LNCL_GETCHAN(E)  (E & 0xff)        // pull out channel
#define DQ_EXTRACT_SS(flags) (((flags) & (LNCL_SS1 | LNCL_SS0)) >> 28)
#define DQ_EXTRACT_DIR(flags) (((flags) & LNCL_INOUT) >> 30)
#define DQ_SS_DIR(ss, dir) (((ss) << 1) | (dir))
```

The configuration flags serve different functions:

`DQ_LNCL_NEXT` - specifies that there is a following channel list entry in the channel list. A channel list entry without this flag set is considered the last one. Advanced and ACB functions add this flag automatically

`DQ_LNCL_INOUT` - specifies whether this is input or output channel for multifunction layers

`DQ_LNCL_SS1` – specifies the subsystem to which the channel belongs. Do not use for single-subsystem layers

`DQ_LNCL_SS0` - specifies the subsystem to which the channel belongs. Do not use for single-subsystem layers

`DQ_LNCL_IRQ` – causes the layer to fire an IRQ upon processing this entry. Required for special real-time cases

`DQ_LNCL_NOWAIT` – causes the layer to temporarily “forget” about the CV clock and start execution of the next channel list entry right after the current one is completed

`DQ_LNCL_SKIP -` prohibits storing the data specified in this channel list entry into the data output FIFO or prohibits advancing the data input FIFO pointer. This flag is used to increase the settling time

`DQ_LNCL_CLK –` causes the channel list machine to wait for the next channel list clock. Normally, the state machine executes the whole channel list on a single CL clock.

`DQ_LNCL_CTR –` perform a pulse on the selected line. This flag is used for synchronization purposes

`DQ_LNCL_WRITE -` write the output to the double-register but do not propagate the physical signal to the output.

`DQ_LNCL_UPDALL` – clock all output channel double-registers to update them simultaneously. This entry is usually used with the `DQ_LNCL_WRITE` entry when the user needs to write data to the output channels sequentially and update them at the same time. In this situation, the user should use the `DQ_LNCL_WRITE` flag for every entry. To update all outputs with previously written values, the `DQ_LNCL_WRITE` flag should be combined with the `DQ_LNCL_UPDALL` flag.

`DQ_LNCL_TSRQ -` insert a timestamp into the output data

`DQ_LNCL_SLOW –` double the settling time for this channel

`DQ_LNCL_DIFF –` acquire the channel in differential mode (rather than single-ended or pseudo-differential)
The channel number occupies the first eight bits of the channel list entry. The maximum number of channels on one device cannot be larger than 256.
Bits [11…8] contain gain information. The number of gains and the gain are specific for every layer type. See *powerdna.h* for layer specific gain macros.

5.2.2 Configuration flags

Configuration flags occupy a 32-bit configuration word. The upper part of the configuration word contains layer-specific flags.

```
// Standard part (lower 16 bits) of layer configuration word
// Please notice that for multiple-subsystem layers one should pass
// multiple configuration uint32s in config_io()
//
#define DQ_LN_TSCOPY     (1L<<18)  // copy timestamp along with the data
#define DQ_LN_MAPPED     (1L<<15)  // For WRRD (DMAP) devices
#define DQ_LN_STREAMING  (1L<<14)  // For RDFIFO devices - stream the FIFO data
                                   // automatically
                                   // For WRFIFO - do NOT send reply to WRFIFO
unless needed
#define DQ_LN_RECYCLE    (1L<<13)  // if there is no data taken/available
                                   // overwrite/reuse data
#define DQ_LN_GETRAW     (1L<<12)  // force layer to return raw unconverted data
#define DQ_LN_TMREN      (1L<<11)  // enable layer periodic timer
#define DQ_LN_IRQEN      (1L<<10)  // enable layer irqs
#define DQ_LN_PTRIGEDGE1 (1L<<9)   // stop trigger edge MSB
#define DQ_LN_PTRIGEDGE0 (1L<<8)   // stop trigger edge: 00 - software, 01 -
rising,
                                   // 02 - falling
#define DQ_LN_STRIGEDGE1 (1L<<7)   // start trigger edge MSB
#define DQ_LN_STRIGEDGE0 (1L<<6)   // start trigger edge: 00 - software,
                                   // 01 - rising, 02 - falling
#define DQ_LN_CVCKSRC1   (1L<<5)   // CV clock source MSB
#define DQ_LN_CVCKSRC0   (1L<<4)   // CV clock source 01 - SW, 10 - HW, 11 -EXT
#define DQ_LN_CLCKSRC1   (1L<<3)   // CL clock source MSB
#define DQ_LN_CLCKSRC0   (1L<<2)   // CL clock source 01 - SW, 10 - HW, 11 -EXT
#define DQ_LN_ACTIVE     (1L<<1)   // "STS" LED status
#define DQ_LN_ENABLED    (1L<<0)   // enable operations
```

`DQ_LN_ACTIVE` is needed to switch on the "STS" LED on CPU layer.

`DQ_LN_ENABLE` enables all operations within the layer

`DQ_LN_CLCKSRC0` selects the internal channel list clock (CL) source as a time base. AI-201 supports the CL clock only where the time between consecutive channel readings is calculated by the rule of maximizing setup time per channel. If you'd like to clock CL, clock from an external clock source such as SYNCx line, set the `DQ_LN_CLCKSRC1` flag as well.

`DQ_LN_CVCKSRC0` selects the internal conversion clock (CV) source as a time base. Setting CV clock allows having an equal time period between conversions of different channels. It is mostly used when the user is interested in a phase shift between different channels.

The user can select either the CL or CV clock as a time base. If both clocks are selected, the CL clock is taken as a time base and the CV clock determines the delay between converting channels (i.e. setting time.)

DQ_LN_STRIGEDGE0, DQ_LN_STRIGEDGE1 define the start trigger edge and source. The source can be either software command or external trigger edge.

DQ_LN_PTRIGEDGE0, DQ_LN_PTRIGEDGE1 define the stop trigger edge and source. The source can be either software command or external trigger edge.

DQ_LN_TSCOPY – copy timestamp at the end of every channel list

DQ_LN_MAPPED – set this flag to declare DMap mode

DQ_LN_STREAMING – set this flag to declare ACB mode

DQ_LN_RECYCLE – this flag affects output operation. If this flag is set and layer does not receive output data, it will recycle old data until new data is available; otherwise the layer will stop at the last value output

DQ_LN_GETRAW – tells the layer to return uncalibrated unconverted data. This flag makes sense only for layers with software calibration (AI-225, for example). Moving calibration and conversion of data to host unloads IOM processor

DQ_LN_TMREN – use a real-time timer to retrieve data from the PowerDNA cube. When this mode is selected, the firmware programs the layer to store one channel list worth of data in the buffer. On a timer tick, the firmware transfers this data from the layer output buffer to the packet. This function is used when the hardware allows only a selected set of update rates, but the user needs something in between. For example, AI-225 can convert data with fixed frequency equal 6.875Hz * 2^n, where n = [0…9]. To receive an exactly 500Hz data stream from this layer one should specify to update this layer upon a timer tick.

DQ_LN_IRQEN – use interrupts to retrieve data from the layer output buffer via packets. This is preferable mode of operation.

5.2.3 EEPROM user area access

Every I/O layer has E^2PROM chip which contains 2048 bytes of layer-specific information.
Model and option numbers identify every layer. The model number is hard-coded inside layer logic and option numbers are stored inside E^2PROM.

E^2PROM is divided into certain access areas (some of them can be missing in different layer types):

```
typedef struct {
    DQEECMNDEVS ee;
    DQCALSET_xxx_ calset;
    DQOPMODEPRM_xxx_ opmodeprm;
    DQINITPRM_xxx_ initprm;
    DQSDOWNPRM_xxx_ sdownprm;
    DQCNAMES_xxx_ cname;
} DEVEEPROM_xxx_, *pDEVEEPROM_xxx_;
```

The first part of the layer E^2PROM is common device information defined as:

```
typedef struct {
    /* header is standard for all devices */
                        /* superuser access */
    uint16  model;      /* device model to verify EEPROM identity */
    uint16  option;     /* device option */
    uint16  total;      /* total EEPROM size - EEPROM read is expensive
*/
                        /* if this field <32 or >2048 read all2048 bytes
*/
    uint32  sernum;     /* serial number - pad to %07d when printing */
    uint32  mfgdate;    /* manufacturing date:  0xmmddyyyy */
                        /* user access */
    uint32  caldate;    /* calibration date:    0xmmddyyyy */
    uint32  calexpd;    /* calibration expired: 0xmmddyyyy */

    /* header is followed by device-specific data structures */
} DQEECMNDEVS, *pDQEECMNDEVS;
```

`CALSET_xxx_` contains layer calibration information. Firmware writes this information automatically upon entering initialization mode.

`OPMODEPRM_xxx_` contains layer parameters for operation mode. For example, AI-201 has the following parameters stored:

```
typedef struct {
    uint32 chlst[AI201_CHAN];   // channel list - full
    uint32 conf;                // control word - layer API flags
    uint32 cvclk;               // CV clock
    uint32 clclk;               // CL clock
    uint32 trig;                // trigger configuration
…
} DQOPMODEPRM_201_, *pDQOPMODEPRM_201_;
```

This structure varies from one major firmware revision to another.
When the firmware switches the layer into operation mode, it processes stored configuration information like it would process configuration parameters received

from host. All working fields in the internal device information structure are filled and the unit is ready to switch into operation mode. By programming the `DQOPMODEPRM` structure ahead of time and storing it into E^2PROM, the user can avoid programming the IOM every time before switching into operation mode.

`INITPRM_xxx_` contains initial I/O directions and output levels. The firmware sets up the direction and the level on every output line on entering initialization state.

`SDOWNPRM_xxx_` contains final I/O directions and output levels. The firmware sets up the direction and the level on every output line on entering shutdown state.

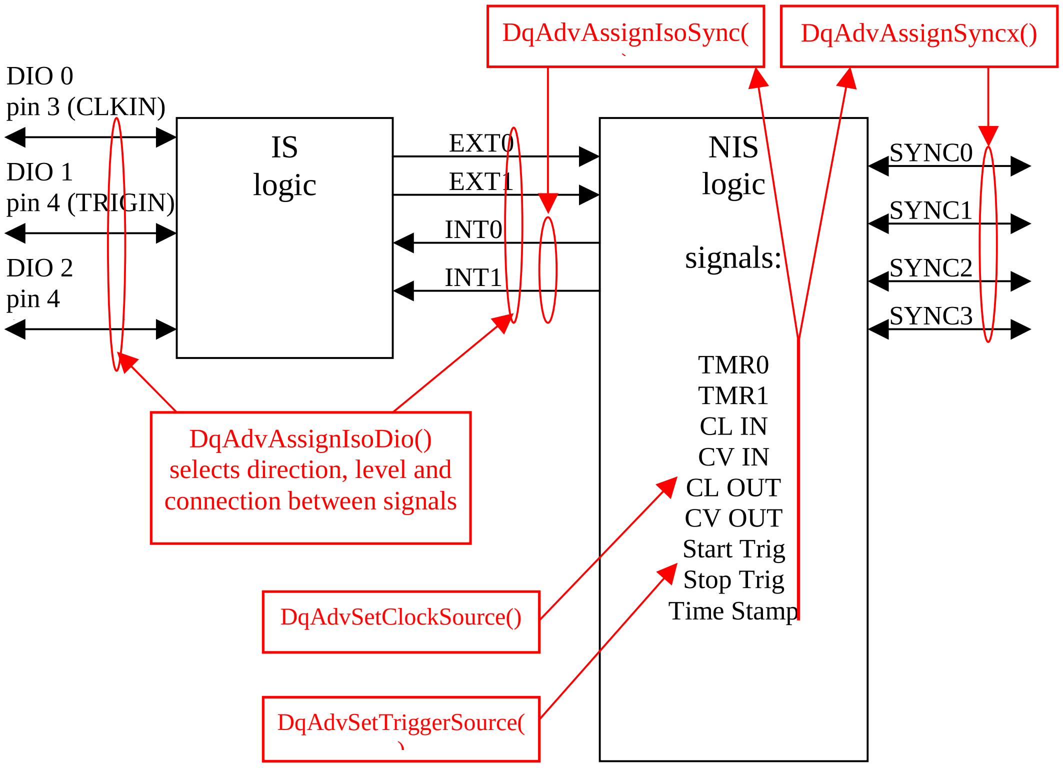
`CNAMES_xxx_` contains channel names. The length of the channel names depends on the layer type. Only 512 bytes are allocated for channel names. Thus, AI-205 layer (four channels) can have channel names as long as 32 characters while DIO-403 channel names (48 channels) cannot be longer then 10 characters.

There is a set of functions written to read, write and store these parameters into E^2PROM. Functions `DqCmdGetParameters()`/`DqCmdSetParameters()` access modal parameters, while `DqCmdSaveParameters()` stores parameters into E^2PROM.

5.2.4 PowerDNA layer signaling

5 Setting up triggering, synchronization and clocking lines

Most PowerDNA layers have the following interconnection diagram:



- *DIO0/CLKIN* – pin 3 on the FJIO1 DB-37 connector. By default this pin is input, connected to the *ISO_EXT0* synchronization line and through this line to the NIS logic
- *DIO1/TRIGIN* – pin 4 on the FJIO1 DB-37 connector. By default this pin is input, connected to the *ISO_EXT1* synchronization line and through this line to the NIS logic
- *DIO2/CLKOUT* – pin 22 on the FJIO1 DB-37 connector. By default this pin is output connected to the *ISO_INT0* line from the NIS logic

PowerDNA API exposes six specially designated functions to control these lines.

- `DqAdvSetClockSource()`
  This function selects external clock source for *CL* (or *CV*) clock. Clock can be selected from internal sources, *EXTx* lines (signals from the isolated side) and *SYNCx* interface signals (inputs)
- `DqAdvSetTriggerSource()`
  This function select external clock source for start and stop trigger. Clock can be selected from internal sources, *EXTx* lines (signals from the isolated side) and *SYNCx* interface signals (inputs)
- `DqAdvAssignIsoDio()`
  This function selects direction and signal assignment for external *DIO* line. *EXT0/1* lines are assigned to *DIO0/1* lines when *DIO* lines are in the input state.
- `DqAdvAssignIsoSync()`
  This function selects signal assignment for *INT* lines. This function allows selecting what signal from isolated side of the layer logic will be assigned to *INTx* lines. Signals can be selected from internal clock sources and *SYNCx* lines.
- `DqAdvAssignSyncx()`
  This function selects signal for each of the *SYNCx* lines. When a *SYNC* line is selected it switches to the output state. All other layers "listen" to this command on the system bus and release that *SYNC* line from use (switch to the input mode). This organization prevents two layers from driving the same line.
- `DqAdvWriteSignalRouting()`
  This function write and activate selected signal routing. This function transfers created configuration to the cube and activates it. Cube sends current synchronization configuration as a reply.

Please note that to take advantage of using external clocks for the layer clock and/or trigger, the source should be selected as external. This means that, in clocking configuration, the following bits should be set up:

- `DQ_LN_CLCKSRC1` – external *CL* clock is selected
- `DQ_LN_STRIGEDGE1` – external start trigger is selected
- `DQ_LN_PTRIGEDGE1` – external stop trigger is selected

If internal sources are selected for those signals, all external signal configurations do not affect layer clocking.

The same interface applies to the CPU layer. The CPU layer has one external input and one output routable to the *SYNCx* interface as well as multiple clocks. It is possible to include an IEEE 1588 implementation with an atomic clock (1us) resolution in the future.

5.3 Register map and description

5.3.1 Register map

All *CTU* registers are located at addresses starting at Base+`DQ_CLI_CTUxS`, where *x* is the CTU number 0-7. I/O FIFO use the standard PowerDNA FIFO locations starting at Base+0x1800.

Register Offset	Name	Description
`DQ_CTU_STR`	*CTU* status register	Counter/timer status register contains status bits related to *CTU* functionality. (read)
`DQ_CTU_CTR`	*CTU* control register	Counter/timer control register contains control bits related to *CTU* functionality. (write)
`DQ_CTU_CCR`	CTU counter control register	Counter/timer Counter Control register – define mode of the operation of the counter and prescaler. (write)
`DQ_CTU_PS`	Prescaler Divider	Program prescaler divider (*PS*) (default value – 0) and read back current value of the prescaler counter. (read/write)
`DQ_CTU_CR`	Count Register	Current value of the count register. (read)
`DQ_CTU_LR`	Program Load Register	Write access set new value of the load register *LR* also copying the same value into the count register *CR*. (write)
`DQ_CTU_IDBC`	Clock Debouncing Register	Program input clock-debouncing register *IDBC*. *CTU* will expect input clock to remain stable for the specified number of 66MHz clocks before processing/ qualifying it. (write)

`DQ_CTU_IDBG`	Gate Debouncing Register	Program input gate-debouncing register *IDBG*. *CTU* will expect input gate line to remain stable for the specified number of 66MHz clocks before processing/ qualifying it. (write)
`DQ_CTU_PC`	Period Count Register	Period count register *PC* is used in a measurement modes when averaging for the multiple periods is required because of the high-speed or unstable nature of the incoming signal. Results of the measurement will be accessible only after specified number of periods on the incoming signal will be detected. (read/write)
`DQ_CTU_CRH` `DQ_CTU_CR0`	CTU Capture Register High	Read: Provide access to the capture register high. Write: Set value of the compare register 0. (read/write)
`DQ_CTU_CRL` `DQ_CTU_CR1`	CTU Capture Register Low	Read: Provide access to the capture register low. Write: Set value of the compare register 0. (read/write)
`DQ_CTU_TBR`	Time Base Register	*TBR* defines time-base divider for the time-based capture modes. Bit 31 (MSB) of the *TBR*. (write)
`DQ_CTU_FCNTI`	Input FIFO Count Register	*CTU* Input FIFO – FIFO0 Count. (read)
`DQ_CTU_FIRQI`	Input FIFO IRQ Level	*CTU* Input FIFO – FIFO0 IRQ. (write)
`DQ_CTU_FDTI`	Input FIFO Data Register	*CTU* Input FIFO – FIFO0 Data In. (write)
`DQ_CTU_FCNTO`	Output FIFO Count Register	*CTU* Output FIFO – FIFO1 Count. (read)
`DQ_CTU_IRQO`	Output FIFO IRQ Level	*CTU* Output FIFO – FIFO1 IRQ level - reserved. (write)

`DQ_CTU_FDTO`	Output FIFO Data Register	*CTU* Output FIFO – FIFO1 Data Out. (read)
`DQ_CTU_ISR`	Interrupt Status Register	*ISR* shows current status of the enabled interrupts. (read)
`DQ_CTU_IER`	Interrupt Enable Register	*IER* is used to specify specific interrupt conditions should generate an interrupt. (write)
`DQ_CTU_ICR`	Interrupt Clear Register	*ICR* allows clearing of "fired" interrupt bits. If interrupt condition persists, interrupt will be fired again. (write)
`DQ_CTU_FDDO`	Output Data FIFO Register	*FDDO* is a reserved register used for the time sequencer version of *CTU* implementation. (write)
`DQ_CTU_TEST0`	Test Register 0	*TEST0* is a reserved test read-only register. In current implementation read from *TEST0* returns 0x01234567. (read)
`DQ_CTU_TEST1`	Test Register 1	*TEST1* is a reserved test read-only register. In current implementation read from *TEST1* returns 0xABCD0123. (read)

The following shows Counter/Timer Units 0-7 registers with 0x80 offset increment representations.

0x2000-0x207C	`DQ_CLI_CTU0S`	*CTU0* I/O registers
0x2080-0x20FC	`DQ_CLI_CTU1S`	*CTU1* I/O registers
0x2100-0x217C	`DQ_CLI_CTU2S`	*CTU2* I/O registers
0x2180-0x21FC	`DQ_CLI_CTU3S`	*CTU3* I/O registers
0x2200-0x227C	`DQ_CLI_CTU4S`	*CTU4* I/O registers
0x2280-0x22FC	`DQ_CLI_CTU5S`	*CTU5* I/O registers
0x2300-0x237C	`DQ_CLI_CTU6S`	*CTU6* I/O registers
0x2380-0x23FC	`DQ_CLI_CTU7S`	*CTU7* I/O registers

5.3.2 Register description

0x2000 RD – CT0_STR – CTU0 status register

The *CTU* Status register is used to report current operational status of the counter/timer unit via dedicated bits for every status condition reported. The Status register mirrors some of the ISR (interrupt status register) bits but it reports current status while ISR reports latched status of the "fired" interrupts

Bit	Name	Description	Reset state
31	`DQ_STR_EN`	When read as 1 indicates that *CR* is enabled in *CT0_CTR* `DQ_CTR_EN` bit.	0
30	`DQ_STR_BUSY`	When read as 1 indicates that *CR* is counting or 0 if current counting operation is complete	0
29	`DQ_STR_CR0L`	When read as 1 indicates that current value of *CR* < *CR0*	0
28	`DQ_STR_CR0GE`	When read as 1 indicates that current value of *CR* >= *CR0*	0
27	`DQ_STR_CR1`	When read as 1 indicates that current value of *CR* >= *CR1*	0
26	`DQ_STR_IN0`	Report current value of direct input pin	0
25	`DQ_STR_GT0`	Report current value of direct gate pin	0
24	`DQ_STR_IN1`	Report current value of de-bounced input pin	0
23	`DQ_STR_GT1`	Report current value of de-bounced gate pin	0
22	`DQ_STR_IHL`	When read as 1 indicates that 1-0 transition was detected on the input pin since last read from *CTx_STR*. This bit will be automatically cleared after each read.	0
23	`DQ_STR_ILH`	When read as 1 indicates that 0-1 transition was detected on the input pin since last read from *CTx_STR*. This bit will be automatically cleared after each read.	0
22	`DQ_STR_GHL`	When read as 1 indicates that 1-0 transition was detected on the gate pin since last read from *CTx_STR*. This bit will be automatically cleared after each read.	0

19	`DQ_STR_GLH`	When read as 1 indicates that 0-1 transition was detected on the gate pin since last read from *CTx_STR*. This bit will be automatically cleared after each read.	0
18	`DQ_STR_OU`	Report current value of output pin	0
17	`DQ_STR_IRQ`	Read as 1 if interrupt was requested	0
16	`DQ_STR_CRH`	Report 1 if data is available in *CRH*	0
15	`DQ_STR_CRL`	Report 1 if data is available in *CRL*	0
14	`DQ_STR_IFE`	Report 1 if input FIFO is empty	0
13	`DQ_STR_IFH`	Report 1 if input FIFO is at least ½ full	0
12	`DQ_STR_IFF`	Report 1 if input FIFO is full	0
11	`DQ_STR_OFE`	Report 1 if output FIFO is empty	0
10	`DQ_STR_OFH`	Report 1 if output FIFO is at least ½ full	0
9	`DQ_STR_OFF`	Report 1 if output FIFO is full	0

0x2000 WR – CT0_CTR – CTU0 control register

The *CTU* Control register is used to set and control some parameters of the operation mode of the counter/timer via specific bits and bit field. Note, the generic interrupt mask/enable/control/status is reported via layer *IER* (0x1C), *IMR*(0x20), *ISR*/*ICR* (0x24) registers. Layer-specific bits are described later in the section. Status conditions which leads to the interrupt request are enabled/disabled via *CTx_CTR* register.

The following are the `DQ_CT0_CTR` Bit descriptions

Bit	Name	Description	Reset state
31	`DQ_CTR_EN`	Enable (1)/Disable (0) counter register. When disabled, *CR* along with pre-scaler and de-bouncer circuitry, freezes it's current operation, which may be re-enabled by writing a one to the `DQ_CTR_EN`, bit.	0
30	`DQ_CTR_IFE`	Input FIFO enable (1) disable(0). Depending on the operation mode, when enabled, fetches one 32-bit word from the input FIFO to the *CR0* at the same time when counter register reloaded with *LR* value	0

29	`DQ_CTR_IFS`	Input FIFO transfer size. Used only when `DQ_CTR_IFE` = 1. 0 - 1 word, 1- 2 words. Defines one (*CR0*) or two words (*CR0*/*CR1*) will be loaded every time when "end-of-count" condition is detected	0
29	`DQ_CTR_IIE`	Enable (1)/Disable (0) inversion of the input pin. Value of the pin is inverted right at the input before debouncing circuitry	0
28	`DQ_CTR_GIE`	Enable (1)/Disable (0) inversion of the gate pin. Value of the pin is inverted right at the input before debouncing circuitry	0
27	`DQ_CTR_OIE`	Enable (1)/Disable (0) inversion of the output pin. When enabled – output pin polarity is inverted at the last stage of creating the output	0
26	`DQ_CTR_OU`	Current value of the output pin in GPIO mode (valid if `DQ_CTR_EN` bit = 0 and `DQ_CTR_GPIO`=1)	0
25	`DQ_CTR_OFE`	Output FIFO – enable (1)/disable(0). Depending on the operation mode, when enabled, copies one or two 32-bit words from the input *CR* or *CRH*/*CRL* into the output FIFO when counter reaches end of the count condition	0
24	`DQ_CTR_CLFI`	If this bit is set during write to *CTR* all input path will be cleared (*CRH*/*CRL* and input FIFO), FIFO will contain 0 samples and *CRH*/*CRL* will be set to 0. Reset input FIFO before initiating any HOST→CTU transfers	
23	`DQ_CTR_CLFO`	If this bit is set during write to *CTR* all output path will be cleared (*CR0*, *CR1*, *LR* and output FIFO), FIFO will contain 0 samples and all registers affected will be set to 0. Reset output FIFO before initiating any CTU→HOST transfers	

22	`DQ_CTR_CLR`	If this bit is set during write to *CTR CTUx* will be reset to the default state, all registers/FIFO will be cleared	
21	`DQ_CTR_GPIO`	If this bit is set, GPIO operation of the "*clkout*" pin is enabled	0

`DQ_CTR_EN`	`DQ_CTR_GPIO`	`DQ_CTR_OU`	*clkout*
0	0	x	remains in a last state
0	1	0	0
0	1	1	1
1	x	x	defined by the current *CTU* mode

20-0		Reserved	0

0x2004 WR – CT0_CCR – CTU0 Counter control register

CTU Counter Control register is used to set current mode for the counter and pre-scaler.

The following are the *CT0_CCR* Bit descriptions.

Bit	Name	Description	Reset state
31	`DQ_CCR_RE`	Enable re-load of the *CR* by the value loaded in *LR* when it reaches end of the count. End of the count is limited by one of the combinations of `DQ_CCR_EC2/1/0` bits	0

30-28	`DQ_CCR_EC2`
	`DQ_CCR_EC1`
	`DQ_CCR_EC0`
		Set end of the count mode:
		(0)000 – `DQ_EM_CR0`, end, when reaches *CR0* (CR=CR0)
		(1)001 – `DQ_EM_CR1`, end, when reaches *CR1* (*CR=CR1*)
		(2)010 – `DQ_EM_FFF`, end, when reaches 0xFFFFFFFF
		(3)011 – `DQ_EM_PC`, end, when X periods of the signal is captured. X is defined via *CTx_PC* (0x2018) register. In width (1/2
			0

		period) measurement mode end, when positive part of the input signal is captured. (4)100 – `DQ_EM_TBR`, end, when time-base counter reaches 0 Note: all other modes are reserved for the future use and will be recognized as a mode 0	

24-27	`DQ_CCR_CRM3`
	`DQ_CCR_CRM2`
	`DQ_CCR_CRM1`
	`DQ_CCR_CRM0`
		Set counter mode:
		(0x0)0000 – `DQ_CM_CT` counter (*CR* acts as a standard count-up counter, 66MHz base clock used as a *PS* source)
		(0x8)1000 – `DQ_CM_ECT` counter (*CR* acts as a standard count-upcounter, debounced *CLKIN* clock used as a *PS* source)
		(0x9)1001 – `DQ_CM_HP` capture ½ period mode (*CR* captures ½ period of the input signal starting from the rising edge of the de-glitched input and copies it into *CRH*).
		(0xA)1010 – `DQ_CM_NP` capture full period (*CR* captures length of the full period, copies positive part of the period into *CRH* and negative (low) into *CRL*, if *CTx_PC* > 0 – continue this process increasing *CRH/CRL* for the length of positive/negative part of every period
		(0xB)1011 – `DQ_CM_QE` quadrature encoder mode
		(0x4)0100 – `DQ_CM_TCT` same as 0x0 but with trigger
		(0xC)1100 – `DQ_CM_TECT` same as 0x8 but with trigger
		(0xD)1101 – `DQ_CM_THP` same as 0x9 but with trigger
		(0xE)1110 – `DQ_CM_TNP` same as 0x9 but with trigger
		
		Note, that all modes, except mode 0 are using debounced *CLKIN* pin as a clock
			0

		source for the pre-scaler. Trigger source (Harware/Software) is selected using `DQ_CCR_TRS` bit	
23	`DQ_CCR_PSG`	Enable(1)/Disable(0) hardware gate on the prescaler. If enabled, GATE input, when positive, enables pre-scaler counter. Note, that `DQ_CTR_EN` bit in *CTR* may be effectively used as a software gate, when `DQ_CCR_PSG` = 0.	0
22	`DQ_CCR_TRS`	Select Hardware(1)/Software(0) trigger source for the triggered modes. Hardware-triggered modes will start at low-high transition on the *GATE* input. In software trigger mode `DQ_CTR_EN` bit in *CTR* should be used as a trigger (`DQ_CTR_EN` will be cleared at the end of the counting operation if *CCR_TRS* bit is cleared and triggered mode is selected `DQ_CM_Txx`)	0
21	`DQ_CCR_ENC`	This bit compliments `DQ_CCR_TRS` bit and works only in a triggered mode – if set (1) enables auto-clear of the `DQ_CTR_EN` bit at the end of the current operation.	

6 Valid EM/CM combinations for non-buffered modes

Refer to the table below for the possible EM/CM combinations (x – valid mode):

	DQ_ EM_ CR0	DQ_ EM_ CR1	DQ_ EM_ FFF	DQ_ EM_ PC	DQ_ EM_ TBR	**Notes**

DQ_CM_CT *DQ_CM_ECT* *DQ_CM_TCT* *DQ_CM_TECT*	x	x	x		x	Use DQ_EM_CR0 for the single-clock pulse generation, DQ_EM_CR1 for *PWM* mode, DQ_EM_FFF for wrap-around counter
DQ_CM_HP *DQ_CM_NP* *DQ_CM_THP* *DQ_CM_TNP*				x		
DQ_CM_QE	x	x	x		x	For continuous non-buffered operation of *QE*, it is recommended to use DQ_EM_TBR mode but disable timebase counter by writing 0x1 to *TBR*

<u>0x2008 WR – CT0_PS – CTU0 Prescaler</u>
Set value of the pre-scaler. Prescaler is a 32-bit count-down counter output of which is used to clock counter register (*CR*). Source for the prescaler is automatically selected based on current value of the *CCR_CRMx* bits. Note, that if pre-scaler is loaded with 0, it will be by-passed and an input signal will be used as an input clock for the count register *CR* (but *GATE* pin if used will still affect the counter).

<u>0x2008 RD – CT0_PS – CTU0 Prescaler Current Value</u>
Read current 32-bit value of the prescaler.

0x200C RD – CT0_LR – CTU0 Load Register
32-bit value, stored in the load register *LR*, will be loaded into the main counter CR at the beginning of each counting cycle.

0x200C RD – CT0_CR – CTU0 Count Register Current Value
Current value of the count register, latched at the time of the read.

0x2010 WR – CT0_IDBC – CTU0 input pin debouncing filter counter register
Program input clock-debouncing register 32-bit register *IDBC*. *CTU0* will expect input clock to remain stable for the specified number of 66MHz clocks before processing/qualifying it.

0x2014 WR – CT0_IDBG – CTU0 gate pin debouncing filter counter register
Program input gate-debouncing register *IDBG*. *CTU0* will expect input gate line to remain stable for the specified number of 66MHz clocks before processing qualifying it.

0x2018 RD – CT0_PC – CTU0 Current value of the period counter register
32-bit current value CTU0 period count register
CT0_PC
Set *CTU0* period count register
Period count register (*PC*) is used in a measurement mode when averaging for the multiple periods it is required because of the high-speed or unstable nature of the incoming signal. Results of the measurement will be accessible only after specified number of periods on the incoming signal will be detected. Start of the period assumed to be a rising edge of the de-bounced input *CLKIN* line.

0x201C RD – CT0_CRH – CTU0 Capture register HIGH
32-bit register is used to store results of the measurements in ½ or N periods measurement modes. In N periods (N is defined by the value stored in the *PC* register) measurement mode provide accumulated number of 66MHz counts during the positive part of all periods measured.

0x201C WR – CT0_CR0 – CTU0 Set value of the Compare Register 0
32-bit compare register zero (*CR0*) is used to define shape of the output signal. In all modes except quadrature encoder and measurement modes, counter register *CR* counts up from the value loaded in *LR* register and output toggles from low to high when *CR*=*CR0*. Depending on the other configuration parameters selected,

counter may continue count, restart itself or stop, when value of the *CR* reaches value stored in *CR0* register. *CR0* may be used in conjunction with *CR1* for the complex PWM waveform generation

0x2020 RD – CT0_CRL – CTU0 Capture register LOW
32-bit register is used to store results of the measurements in N periods measurement modes. In N periods, (N is defined by the value stored in the PC register) measurement mode provides accumulated number of 66MHz counts during the negative (low) part of all periods measured.

0x2020 WR – CT0_CR1 – CTU0 Set value of the Compare Register 1
32-bit compare register one (*CR1*) is used to define shape of the output signal. In all modes except quadrature encoder and measurement modes, counter register *CR* counts up from the value loaded in *LR* register and output toggles from low to high when *CR*=*CR0*, then output stay high until *CR0*<=*CR*<=*CR1*. Depending on the other configuration parameters selected, counter may continue count, restart itself or stop, when value of the *CR* reaches value stored in *CR1* register. *CR1* may be used in conjunction with *CR0* for the complex PWM waveform generation

0x2024 WR – CT0_TBR – CTU0 Time-base divider register
32-bit *TBR* (write-only) register defines time-base divider for the time-based capture modes.

7 FIFO access

0x1800/0x2028 RD – CT0_FCNTI – CTU0 Input FIFO Count Register
9-bit, LSB valid, return number of samples available (written from the host to layer) in the input FIFO of the *CTU0*.

0x1808/0x2030 WR – CT0_FDTI – CTU0 Input FIFO Data Input Register
32-bit write-only register for the input FIFO.

0x1810/0x2034 RD – CT0_FCNTO – CTU0 Output FIFO Count Register
9-bit, LSB valid, return number of samples available in the output FIFO of the *CTU0*.

0x1818/0x203C RD – CT0_FDTO – CTU0 Output FIFO Data Input Register
32-bit read-only register for the input FIFO.

0x2040 WR – CT0_IER – CTU0 Interrupt Enable register

Interrupt generation unit in every *CTU* is similar to the *IGU* in PDNA CLI logic except it does not have interrupt mask register for the simpler operation. Interrupt from any of the available sources, if enabled, latched in *ISR* 0x2040+RD (interrupt Status register) and forces IRQ request into logic HIGH state. IRQ line remains in HIGH state until all IRQ requests are cleared via *ICR* 0x2044+WR (Interrupt clear register).

The *IER* register is a bit field where each bit enables one interrupt source.

The following are the *CT0_IER* Bit descriptions.

Bit	Name	Description	Reset state
31	`DQ_IR_CPT`	Request interrupt if counter complete current operation	0
30	`DQ_IR_CR0L`	Request interrupt if current value of *CR* < *CR0*	0
29	`DQ_IR_CR0GE`	Request interrupt if current value of *CR* >= *CR0*	0
28	`DQ_IR_CR1`	Request interrupt if current value of *CR* >= *CR1*	0
27	`DQ_IR_LHI`	Request interrupt if low-high transition was detected on the input pin (deglitched)	0
26	`DQ_IR_LHG`	Request interrupt if low-high transition was detected on the gate pin (deglitched)	0
25	`DQ_IR_HLI`	Request interrupt if high-low transition was detected on the input pin (deglitched)	0
24	`DQ_IR_HLG`	Request interrupt if high-low transition was detected on the gate pin (deglitched)	0
23	`DQ_IR_CRH`	Request interrupt if data is available in *CRH*	0
22	`DQ_IR_CRL`	Request interrupt if data is available in *CRL*	0

21	`DQ_IR_IFE`	Request interrupt if input FIFO is empty	0
20	`DQ_IR_IFH`	Request interrupt if input FIFO is at least ½ full	0
19	`DQ_IR_IFF`	Request interrupt if input FIFO is full	0
18	`DQ_IR_OFE`	Request interrupt if output FIFO is empty	0
17	`DQ_IR_OFH`	Request interrupt if output FIFO is at least ½ full	0
16	`DQ_IR_OFF`	Request interrupt if output FIFO is full	0
15-0		Reserved	0

0x2040 RD – CT0_ISR – CTU0 Interrupt Status register

This register should be used to define source of the interrupt from the *CTU*. It will show "1" in the bits that are the source for the interrupt. The *ISR* keeps it's value until cleared by a write to the *ICR* or by system reset.

CT0_ISR Bit description

The *ISR* bits match *IER.*

0x2044 RD – CT0_ICR – CTU0 Interrupt Clear register

Writing one to any of the bits in ICR will clear matching bit in ISR thus clearing interrupt request based on that bit. Note, that if interrupt condition is still exist and enabled – it will be "fired" again immediately.

CT0_ICR Bit description

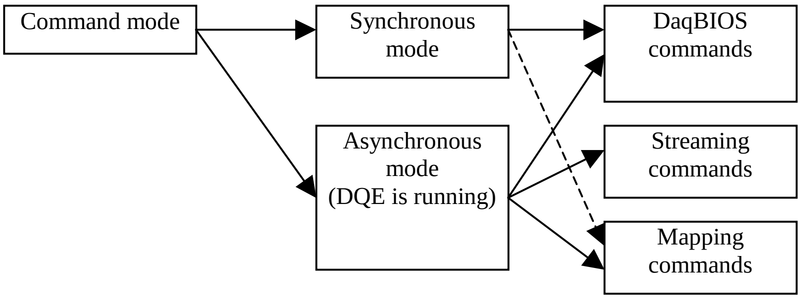
The *ICR* bits match *IER/ISR.*

6 Three ways to communicate with IOM

6.1 Synchronous and asynchronous protocols

PowerDNA API provides three ways of communication with the PowerDNA cube:

1. DaqBIOS Command API (synchronous)
2. Buffered I/O (asynchronous)
3. Mapped I/O (asynchronous)



In synchronous mode, the host sends a request and waits for a reply, then it sends another command.
In asynchronous mode, the host sends requests on ticks of the timebase timer. Asynchronous mode takes care of re-requests in the case of packet loss or network collision. In asynchronous mode, the user can work the same way as in synchronous by sending request after request and processing packets himself. However, we encourage the user to use asynchronous mode for streaming or data mapping and designing his application around this paradigm.

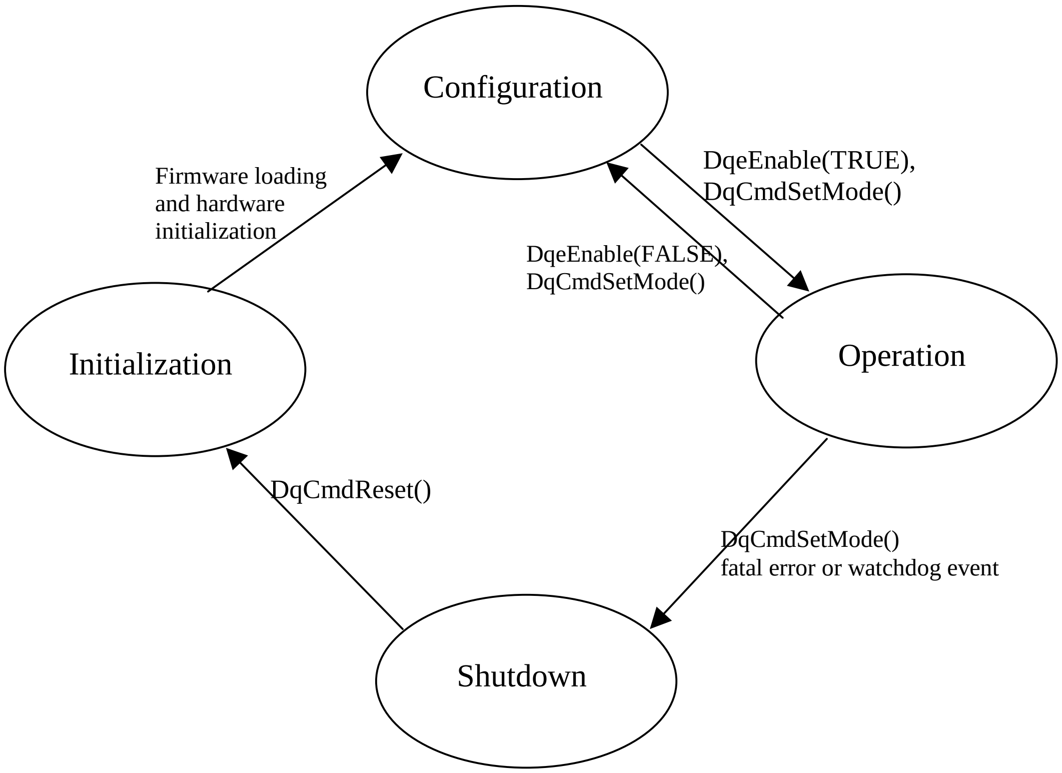
Asynchronous mode is inherently soft-real-time because collisions on the network cannot be predicted and, therefore, cannot be avoided.
For real-time response under the control of a real-time OS, the user can perform mapping using synchronous mode commands or use the FIFO interface to retrieve/send the stream of data. Synchronous mode does not have error correction and data-flow control built-in, and the user is required to perform this task himself.

All three APIs can be used to communicate with the same IOM but not at the same time on one layer. Once the PowerDNA cube is switched to asynchronous mode, do not issue synchronous commands to avoid interfering with PowerDNA cube configuration and timing set up for asynchronous mode.

6.2 IOM Modes

Once the user has started up IOM, the PowerDNA cube can operate in four main modes of operation:

- Initialization
- Configuration
- Operation
- Shutdown


Configuration
Firmware loading and hardware initialization
DqeEnable(TRUE), DqCmdSetMode()
DqeEnable(FALSE), DqCmdSetMode()
Initialization
Operation
DqCmdReset()
DqCmdSetMode() fatal error or watchdog event
Shutdown

The host software controls the switching from state to state via API function calls:

IOM modes of operation

In each of the modes, it’s important to understand what happens on both the host computer and on the IOM.

6.2.1 Initialization mode

Host behavior

1. Does nothing

IOM behavior

Upon power-up, each IOM:

1. Copies user firmware into RAM and starts execution
2. Retrieves data from the parameter sector of the flash memory and performs initialization accordingly
3. Initializes Fast Ethernet controller and TCP/IP stack
4. Initializes chip selects and creates memory map
5. Finds all devices attached to the bus (up to sixteen), assigns addresses and interrupts
6. Switches all devices into initialization mode by calling proper entry point of each device driver
7. Reads EEPROM data and performs an initialization sequence. It resets each device and programs the state of the devices stored in EEPROM. It also programs the Transfer/Channel Lists. The driver also allocates all required memory structures and buffers and attaches them to the device object allocated by the initialization routine
8. calls the driver entry function to switch every device into Configuration mode. In this mode device drivers write calibration values into digital calibration subsystem
9. Checks if <autorun> field is 1 then firmware switches device into Operation mode. Otherwise it switches to Configuration mode

The IOM keeps initialization data in the PARAM sector of the Flash memory (Flash device sector SA1) and EEPROM of every device layer.

6.2.2 Configuration mode

Configuration mode is designed to perform synchronous operations and to set up parameters for asynchronous operations. In this mode, the user can set up operational parameters and the Channel List.

Host behavior

The host can send and receive single commands.

IOM behavior

Each IOM executes each DaqBIOS command upon its arrival. At this point, an IOM doesn't perform continuous acquisition or prepare data for I/O. In this mode, the IOM is capable of performing single-scan reads and writes. The first of those reads (or writes) programs the channel list of the layer for one scan acquisition and starts low-rate data acquisition. All access to these data is organized using `DQCMD_IOCTL` command which resolves into `DqAdv`*`xxx`*`Read()` and `DqAdv`*`xxx`*`Write()` at the host level, where *xxx* is the layer model.

In configuration mode, the user can overwrite the current default parameters of configuration, clock settings, channel/transfer lists stored in EEPROM. All configuration-changing commands apply to current control set of a layer and do not affect EEPROM memory. This current control set of parameters is used to program the layer when the host switches it into operation mode.

6.2.3 Operation mode

Host behavior

If DQE is running, the host continuously sends and/or receives a stream of data and controls data flow.

IOM behavior

Each IOM performs continuous acquisition as defined in the transfer/channel lists. Once the layer is switched into operating mode, it waits for a trigger to start operation. In the case of streaming operations, a trigger can be either an external event or a software command. In the case of data mapping, the layer starts I/O immediately after switching into operating mode.

6.2.4 Shutdown mode

Host behavior

The host sends a shutdown request to all IOMs under its control. Another way to enter shutdown mode is to program a watchdog clock on an IOM and allow it to expire. A watchdog timer clock resets every time an IOM doesn’t receive packets from the host for a specified period of time. Please note that when the IOM layer is set up to perform input streaming, operation data is sent from the IOM side and the host intervenes on an error only. In this situation, the user should enable heartbeat operation before engaging the watchdog.

Shutdown mode is most important for output devices to switch its output into a predictable state (voltage level, tri-stated, etc.)

IOM behavior

When an IOM receives a request for Shutdown mode, it executes a programmed sequence. It brings all I/Os to a safe state which is stored in shutdown parameter area of the layer EEPROM.

An IOM also can switch itself into Shutdown state on the following events:

- The watchdog timer fires
- Communication with the host is lost and cannot be resumed in a specified period of time

One of the ways to ensure fail-safe operations is to set up <autorun> in the <fwct> parameter to 1. Then, when the watchdog fires, it resets the IOM completely. Then the IOM goes, via initialization and configuration mode, into operation mode without intervention from the host. In this circumstance, if, for example, the host was engaged in data mapping operations and one IOM fails, the communication will be lost during the time of booting up this IOM and switching back into operating mode. Once in operation mode, the IOM becomes accessible again with the parameters stored in EEPROM operation mode section.

7 How DaqBIOS protocol works

7.1 DaqBIOS packet structure

The DaqBIOS (DQ) protocol relies on the Ethernet protocol to be transferred. Current implementation of the IOM firmware allows exchanging DaqBIOS packets over raw Ethernet packets and over UDP packets. Library implementation under Microsoft Windows tm does not have an option of using raw Ethernet packets.

Ethernet header (14 bytes)	IP header (20 bytes)	UDP header (8 bytes)	DQ header (16 bytes)	DQ data (6-514	Ethernet CRC (4 bytes)

DaqBIOS packet over UDP packet

Ethernet header (14 bytes)	DQ header (16 bytes)	DQ data (34-542 bytes)	Ethernet CRC (4 bytes)

DaqBIOS packet over raw Ethernet packet

The DaqBIOS protocol relies on a simple concept of acknowledging every packet sent from host to IOM.

The DaqBIOS packet header has following fields:

```
typedef struct {
    uint32 dqProlog;       /* const 0xBABAFACA */
    uint16 dqTStamp;       /* 16-bit timestamp */
    uint16 dqCounter;      /* Retry counter + bitfields */
    uint32 dqCommand;      /* DaqBIOS command */
    uint32 rqId;           /* Request ID - sent from host, mirrored */
    uint8  dqData[];       /* Data – 0 to 514 bytes */
} DQPKT, * pDQPKT;
```

`dqProlog` is always `0xBABAFACA` for revision 2 of the DQ-TS protocol. The DQ-VT protocol available earlier is no longer supported in R2. Instead, we use flow-control and error-correction protocols. The only exception is when you can send a packet with `0xBABAFAC2` as a prolog. In this case, the IOM replies with a proper Prolog and protocol version supported in `dqTStamp`.

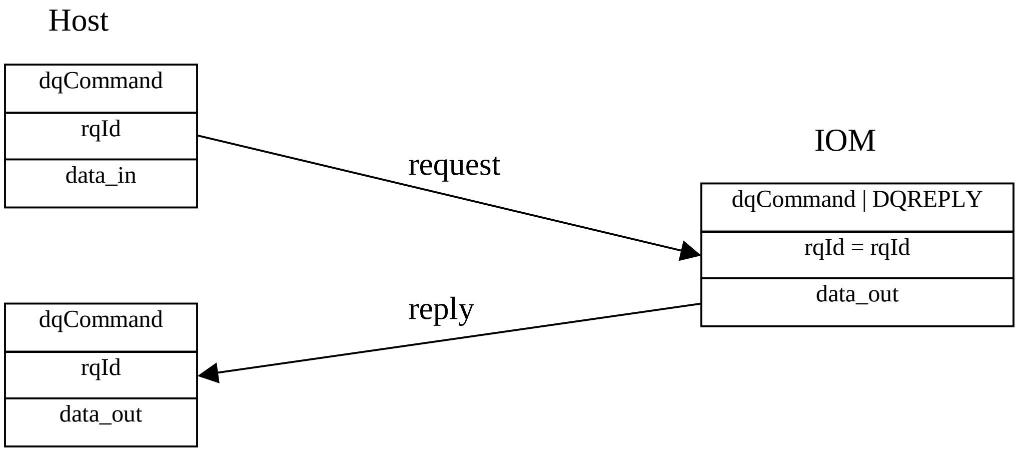
`dqTStamp` is a field used for time synchronization between the IOM and the host.

`dqCounter` is used for flow-control between the host and the IOM. The counter starts from one and continues up to 65535, then wraps around.
`dqCommand` is used to specify the command to execute when sent from the host to the IOM. The host replies with the command executed and with any error flags set. If the IOM processes the command successfully, it replies with the requested command and the `DQREPLY` (0x1000) flag. If the host sends a command with a `DQNOREPLY` (0x2000) flag, the IOM doesn't send a reply packet.
The following errors located in the upper 16 bits of `dqCommand` are sent in `dqCommand` field:

```
/* Masks to extract DQERR_... from command code */
#define DQERR_MASK      0xFFFF0000
#define DQNOERR_MASK    0x0000FFFF

/* The first nybble indicates how the next three nybbles should be interpreted */
#define DQERR_NYBMASK   0xF0000000 /* general error/status mask */
#define DQERR_MULTFAIL  0x80000000 /* high bit - multiple bits indicate
error/status */
#define DQERR_SINGFAIL  0x90000000 /* low bit in first nybble - single
error/status */

#define DQERR_BITS      0x0FFF0000 /* error/status bits or value extracted from
here */

/* multiple errors - inclusive or-ed with dqCommand -- high bit set */
#define DQERR_GENFAIL   0xF0000000 /* general error/status mask */
#define DQERR_OVRFLW    0x80010000 /* Data extraction too slow - data overflow */
#define DQERR_STARTED   0x80020000 /* Start trigger is received */
#define DQERR_STOPPED   0x80020000 /* Stop trigger is received */

/* single errors/status - not inclusive or-ed bit 0x10000000 set */
#define DQERR_EXEC      0x90010000 /* exception on command execution */
#define DQERR_NOMORE    0x90020000 /* no more data is available */
#define DQERR_MOREDATA  0x90030000 /* more data is available */
#define DQERR_TOOOLD    0x90040000 /* request is too old (RDFIFO) */
#define DQERR_INVREQ    0x90050000 /* Invalid request number (RDFIFO) */
#define DQERR_NIMP      0x90060000 /* DQ not implemented or unknown command */

/*
** The following is reuse of the previous code
** in the different direction: host->IOM
** It means that there was no reply to one
** of the previous packets of the same type
** Made especially for RDALL, WRRD and RDFIFO
** commands.
*/
#define DQERR_OPS       0x90070000 /* IOM is in operation state */
#define DQERR_PARAM     0x90080000 /* Device cannot complete request with
specified parameters */

/* network errors */
#define DQERR_RCV       0x90090000 /* packet receive error */
#define DQERR_SND       0x900A0000 /* packet send error */
```

`rqId` – request ID. Every time the host sends a packet to IOM, it is accompanied with a new request ID. The Request ID serves to specify what request the reply belongs to when request/reply pairs are overlapped. `RqId` is used under the control of DQE only.

In synchronous operating mode, commands are sent and replies are received. The following picture depicts how the host and the IOM exchange packets under the DaqBIOS protocol:



7.2 DaqBIOS protocol versions

To recognize what version of the DaqBIOS protocol the PowerDNA cube supports, the host should send a command with `dqProlog` set to`0xBABAFAC2`. The IOM will reply with the proper prolog and the DaqBIOS protocol version in the `dqTStamp` field and the firmware version in the next four bytes. This sub-protocol allows the host to recognize what version of the firmware is running on the PowerDNA cube and what version of protocol it supports.

7.3 Host and IOM data representation

Data on the IOM as well as in the network packets are represented in big-endian format. Data on the PC platform are rendered in little-endian format. Thus, to ensure proper data representation the user should convert data from network to host format and back.

7.4 Soft and hard real-time

We address real-time performance as soft-real-time when timing deadlines are achieved almost every time. However, soft-real-time cannot guarantee meeting a deadline in all instances. The majority of general-purpose OSes (Microsoft Windows, Linux, etc.) are soft-real-time with better or worse probability of missing a deadline.

Hard-real-time performance guarantees that no one deadline is missed. Hard-real-time OSes have specially designed schedulers that preempt any ongoing operation when real-time code has to be executed. QNX and RTLinux are examples of hard-real-time OSes.

7.4.1 Implementation details

Hard real-time response is achievable only under control of hard-real-time OSes (QNX, for example) or general-purpose OSes with real-time extensions (RTLinux, RTAI Linux.) Real-time OSes are capable of sending DaqBIOS commands to the host without missing deadlines (using DQE). This avoids network collisions completely. Two sets of commands are available for real-time operations: DaqBIOS commands and data mapping commands. Streaming cannot be made real-time because its timing cannot be controlled from the host side.

If streaming is required under a real-time system, you can implement streaming in FIFO mode rather than streaming mode. FIFO mode assumes that the host sends a request to retrieve data from the IOM side every now and then. This way, the real-time application is responsible for retrieving data on time.

7.4.2 Immediate and pending commands

The firmware processes some commands immediately in the network interrupt vector. Other commands are scheduled and executed by firmware in the pending command thread. A vast majority of DaqBIOS commands are immediate commands. See the *PowerDNA API Reference Manual* for details. Firmware running on a CM-1 layer sends replies within 200-400µs. Commands that include waiting for some hardware events to happen are implemented as pending commands. They include IOCTL calls, setting/getting/saving parameters, and receiving layer capabilities information. The time for pending command execution varies and the user should adjust the timeout prior to calling these commands appropriately.

7.5 DaqBIOS & Network Security

The PowerDNA Cube may be connected to the Internet, posing virtually no risk to the network it is hosted on. Several features make the PowerDNA Cube next to invulnerable for external attack, in descending order:

(1) The PowerDNA Cube has only one UDP open port.
By default this port is 6334 - falling in the IANA *unassigned port range 6323-6342*. Default security hole scanners will either skip UDP scanning, or skip scans of this range, expecting no useful protocols to run in this range.

(2) The only protocol running on the cube is DaqBIOS – an unpublished protocol with no known exploits.
If UDP port 6334 is discovered, it is unusable by anyone who does not understand the protocol.

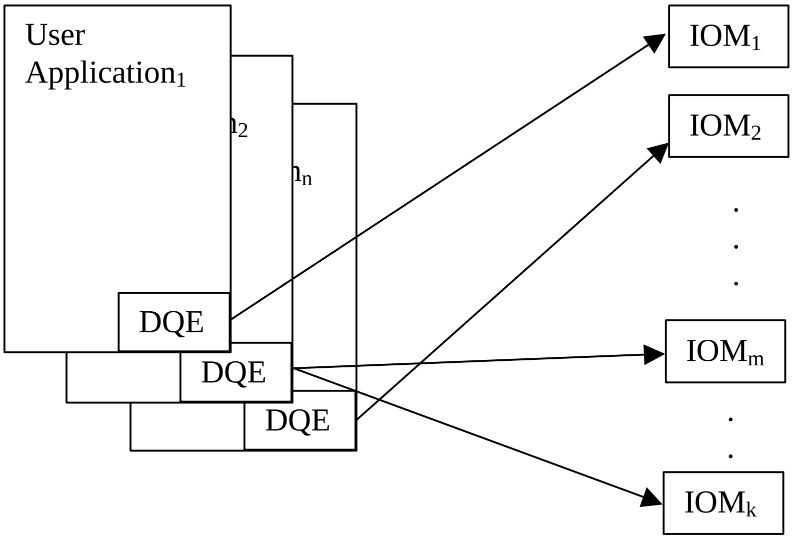
(3) Commands over the network that involve a change to the IOM memory or settings require a password. Any command that changes internal state of the cube requires user password to be supplied. The password is stored in the encoded NVRAM area of RTC chip. Any command that changes non-volatile memory requires super-user password. Password is supplied over DQ protocol.

(4) To prevent disruption of the experiment, the cube has the option to be locked onto an IP/port pair. For compatibility, locking/unlocking is disabled by default. When the locking option is enabled and the host PC establishes communication with the cube, the cube locks on to the host's IP/port pair and will listen for commands only from the locked host – until the host unlocks/releases the cube. Other PCs can only request cube configuration and status requests (e.g. IOM_25431 with AI-201 layer in slot 0 is currently in Locked state).

Finally, note that the PowerDNA Cube has no known exploitable daemons (e.g. Ms-IIS for http, ftp, etc.)

8 DaqBIOS Engine

The DaqBIOS Engine (DQE) is organized as a PowerDNA shared library with which a user application is linked. It is a set of functions and data structures, implementing the DaqBIOS data acquisition protocol. DQE provides all functions necessary to interact with IOMs over the network.
DQE functions are executed within the user process; however, DQE may create additional execution threads for its purposes. Different user applications can use DQE simultaneously. Every process gets its own copy of DQE. DQE implements interlock mechanisms preventing using of a single IOM by two processes and a single layer in exclusive modes.
DQE is used to simplify PowerDNA programming and shift data contingency and buffering responsibility from a user application to the library.



User Application/DQE/IOM Interaction.

In the above picture, one user application may interact with more than one IOM. The opposite is not true.

8.1 Basic architecture

DaqBIOS Engine consists of the following parts:

- **Sending thread/periodic task (multimedia timer callback under Windows)**
  This piece of code periodically wakes up and checks the *command queue*

(CQ) of each IOM accessed by the process. It sends one or more commands per IOM per execution cycle and marks it as “waiting for response” so that it isn’t sent the next time. See also *command queue* entry below. There is a single sending thread in every DQE

Receiving thread
There is exactly one receiving thread per each IOM. This thread listens to the IOM, receives packets, and routes them to the input buffers according to the IOM’s *command queue*. When a packet arrives from the IOM, receiving thread looks up the corresponding entry in the command queue then relocates packet to the ring buffer. If there is no corresponding CQ entry, the packet is discarded. If there is any callback associated with the entry, the receiving thread calls it with the specified parameter.

- **IOM table**
  The IOM table is a static array inside library and is common to all processes. It contains information about all active IOMs being contacted from this host. It includes the list of layers and their options, the processes, which are working with them (one process per IOM) and some additional control information. The IOM table access is often made from a critical section.
- **Command queue**
  There is exactly one command queue per IOM. It is a double-linked list that keeps the descriptions (also called command queue entries) of all commands to be sent and all replies to be received to/from the corresponding IOM. The entries are parsed with the *sending thread* and later used by the *receiving thread*. They are put into the queue by `DqSendPkt()` and other DQE calls. The results (after the packets arrive) are used by `DqRecvPkt()` calls or DQE callbacks, specified in the command queue entry.
- **Buffer Control Block**
  This structure contains control information about *Advanced Circular Buffer (ACB)* or *Data Map (DMap)*, such as device, subsystem, transfer list, expected byte rate, update period etc.
- **Reader and Writer threads**
  Reader and writer threads provide transfer of data to and from the packet ring buffer to the ACB or DMap. They are responsible for calling proper data conversion routine depending on the layer type and data format selected. They are also responsible for error correction.
- **Advanced Circular Buffer, Data Map**
  These are the data exchangers between the user application and FIFO devices (for ACB) or groups of snapshot devices (for DMap) on IOM.

8.2 Threads and function

Every instance of DQE has one ***sending*** and one ***receiving*** thread. When process allocates an ACB or a DMap, DQE starts two additional threads. One of them is called ***writer*** thread and another one ***reader*** thread. The purpose of these threads is to transfer data from the ACB to the ring buffer for output and from the ring buffer to the ACB for input. The sending or receiving thread wakes the threads up when data needs to be transferred to/from the ring buffer.

8.3 IOM data retrieval and data conversion

The reader and write threads call a conversion routine that converts data from the raw format represented in the ring buffer into floating point representation of volts or other engineering units. If conversion parameters (offset and coefficient) weren’t supplied upon creation of ACB or DMap then the data conversion routine converts raw data into native representation – Volts.

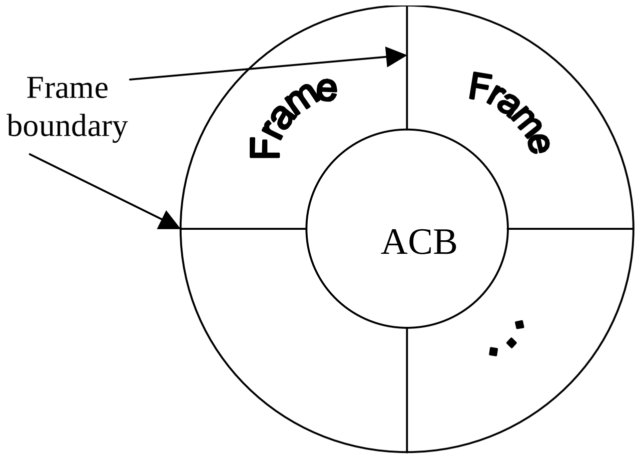
9 Real-time operations with IOM

This section discusses in details how to perform data mapping and streaming under control of a real-time OS. The reason we dedicated a separate section for this operation is that writing a real-time code can be done more efficiently without using the DQE. That's why we will talk about programming streaming and data mapping operations at low-level in this section.

9.1 ACB structure and function

9.1.1 Advanced Circular Buffer Structure

Advanced Circular Buffer (ACB) is a regular FIFO represented as a circular buffer. What you do with the data once they arrive in host memory can also have a major impact on system performance. The PowerDNA software uses a concept of an Advanced Circular Buffer. When combined with applications tuned to take advantage of this flexible buffering mechanism, the system as a whole runs much more efficiently.



ACB and Ring Buffer overall structure

Once an acquisition is started, DQE stores data into the buffer at a known point (called the head), while the application generally reads data at another position (known as the tail). Both operations occur asynchronously and can run at different

rates. However, you can synchronize them by either timer notification or by a DQE event.
To be able to issue a notification to the user application upon receipt of a specific sample or when incoming data reach a scan-count boundary, DQE segments the buffer into frames. Whenever incoming (or outgoing) data crosses a frame boundary, DQE sends an event to the application. If multi-channel acquisition is performed, the frame size should be a multiple of the scan size to keeps pointer arithmetic from becoming unnecessarily complex.

With the ACB, three modes of operation are possible depending on the action taken when the end of the buffer is reached or if the buffer head catches up with the tail.

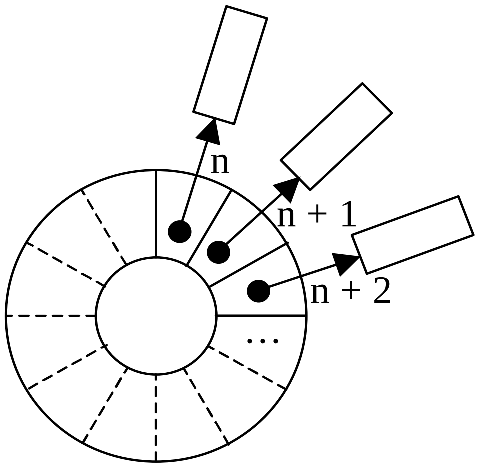
• In **Single Buffer mode**, acquisition stops when DQE reaches the buffer's end. The user application can access the buffer and process data during acquisition or wait until the buffer is full. This approach is appropriate when you're not acquiring data in a continuous stream, and it resembles the way a digital scope operates.

• In **Circular Buffer mode** the head and tail each wrap to the buffer start when they reach the end. If the head catches up to the tail pointer, the buffer is considered full and acquisition stops. This mode is useful in applications that must acquire data with no sample loss. Data acquisition continues until either a predefined trigger condition or the application stops DQE. If the app can't keep up with the acquisition process and the buffer overflows, the driver halts the acquisition and reports an error condition.

• **Recycled mode** resembles Circular Buffer mode except that when the head catches up with the tail pointer, it doesn’t stop but instead overwrites the oldest scans with the new incoming scans. As the buffer fills up, DQE is free to recycle frames, automatically incrementing the buffer tail. This buffer-space recycling occurs irrespective of whether or not the application reads the data. In this mode a buffer overflow never occurs. It's best for applications that monitor acquired signals at periodic intervals. The task might require that the system digitize signals at a high rate but not need to process every sample. Also, an application might need only the latest block of samples.

When the buffer is used for output, the user should fill at least two frames before starting output. Every time a frame becomes empty and ready to accept new data, the DQE triggers an event to the application.

While the ACB might seem a departure from single and double-buffer schemes you'd see on most other data acquisition systems, it's actually a superset of them. In Single Buffer mode, the ACB behaves like a single buffer. Configured as Circular Buffer with two frames, it behaves as a double buffer. With multiple frames, the ACB can function in algorithms designed for buffer queues. The only limitation, which results in a more efficient performance, is that the logical buffers in the queues can't be dynamically allocated or freed and their order is fixed.



Packet Ring Buffer

The Ethernet UDP protocol used to transfer data is connectionless and unreliable. Older packets could come first while new packet may never arrive. The ACB assumes that the data comes sequentially without gaps between scans. To accommodate sequential nature of data stream with the packet nature of Ethernet, DQE implements an intermediate buffer – Packet Ring Buffer (PRB).

PRB is a non-contiguous ring buffer intended for data loss recovery. FIFO devices on the IOM send their data to the host in sequentially numbered packets (using `dqCounter` field of DaqBIOS command header). These numbers vary from 0x1 to 0xFFFF and then wrap around (skipping 0). Such numbering allows

DQE to notice when a packet is missing – if we receive a higher-numbered packet than expected (in the picture above, if the last packet number was `n` and we've just received one numbered `n+2`, we know that the packet `n+1` is missing). Since the receiving buffer is non-contiguous, we just put the newly arrived packet into the buffer, which was bound to receive it anyway and send a specific request for the missing one. When it finally arrives, we will just put it in its place and will be able to copy all data into the contiguous ACB in their proper order.
A thread transfers data from the ring buffer into ACB when contiguous chunks of data become available. The data request routine (`DqGetACBScans()`) also performs additional transfers if there is a chunk of contiguous data available at the moment of execution.

9.1.2 Advanced Circular Buffer Function

When the writer thread writes converted data from the ring buffer into ACB, it checks for frame and buffer boundaries as well as for error conditions. If those conditions exist and the user application has signed up to receive events upon one of these conditions, the writer thread sets up an event (synchronization primitive.) Upon receiving an event, the user application can check what condition caused the event to fire.
An ACB generates the following events:

- **`DQ_eDataAvailable`** is generated when writer thread transfers any data from the ring buffer to ACB. In case of output operation, `DQ_eDataAvailable` is set when there is a place in the buffer to put new data.
- **`DQ_eFrameDone`** is set when incoming data crosses a frame boundary -- when the writer thread has contiguous data in the ring buffer and transfer it into ACB. In the case of output operation, the reader thread takes data from the ACB, converts and puts it into the output ring buffer. Thus, at the beginning of the output operation, the `DQ_eFrameDone` event is set quite often while data is transferred from ACB to the empty ring buffer.
- **`DQ_eBufferDone`** is set in Single mode when buffer becomes full on input or empty on output. Normally `DQ_eBufferDone` is accompanied with `DQ_eStopped` flag.
- **`DQ_ePacketLost`** is set when one or more packets are lost and unrecoverable. In case of input, the DQE tries to request the missing packet for a defined number of times. If this effort fails,

DQ_ePacketLost flag is set. The DQE fills the place allocated for missing packet in the ring buffer with zeroes or with a user-supplied pattern. Then it releases this packet to the write thread. Similar processing happens when the IOM replies to the host with DQERR_TOOOLD flag set. DQERR_TOOOLD means that the requested packet of data is already overwritten with the new data.
However, if the IOM locates the missing packet, it tries to request this packet from host for defined number of times. If the packet is still missing, the IOM sends a message packet to host with a DQERR_LOST flag. This means that IOM cannot recover the missing packet and should skip this packet in the output ring buffer.

- **DQ_ePacketOOB (**packet is out of bounds**)** is set when a packet received by host is so far off that the host cannot insert this packet into the ring buffer. In the similar case for output, when the IOM cannot insert a packet of data received from host into its ring buffer, the IOM replies with DQERR_TOOOLD flag.

9.1.3 DaqBIOS commands used for communications with IOM

The main DaqBIOS command used to transfer data to and from the IOM is DQCMD_RDFIFO (for input) and DQCMD_WRFIFO (for output). When the IOM is programmed for streaming --DQ_LN_STREAMING flag is set in layer configuration – the IOM starts streaming data packets without requests from host. Every packet can contain up to 510 bytes of data. At first, DQE sends a single DQCMD_RDFIFO command to set up initial dqCounter and RqId (request ID) values in DaqBIOS packet header. If the IOM does not reply to this packet, DQE repeats this packet for the number of retries defined. Every packet sent from IOM has the same RqId field in DaqBIOS header equal to the first requested packet. The IOM increments the dqCounter field for each consecutive packet. The DQE can send packets with DQERR_NOMORE and DQERR_MOREDATA flags set in The DqCommand field to stop and restart the data stream from the IOM.

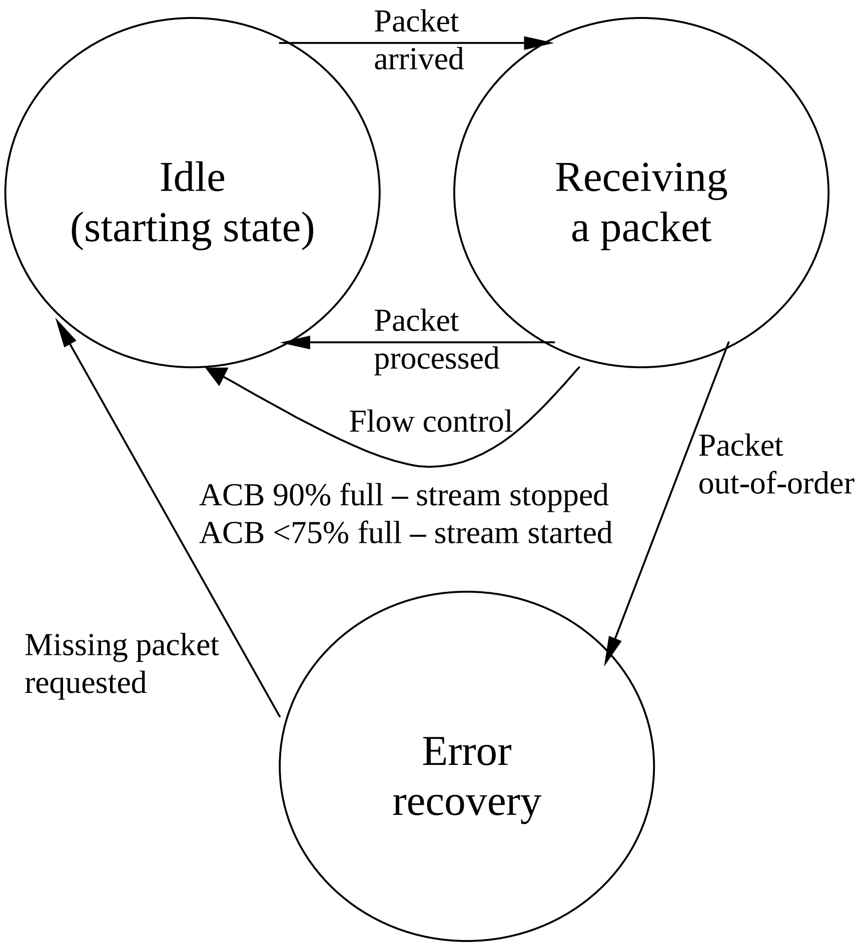
To stream data in the opposite direction (from host to IOM) the DQE configures the IOM layer with the same DQ_LN_STREAMING flag set. Then, the DQE starts sending output data using the DQCMD_WRFIFO command. It numbers every outgoing packet incrementally. If the IOM can accept the packet sent, it doesn't reply. In the case the IOM FIFO becomes 75% (by default) full. The IOM sends a DQCMD_WRFIFO reply with the DQERR_NOMORE error flag. The DQE stops sending packets until it receives another `DQCMD_WRFIFO` packet from the IOM with the `DQERR_MOREDATA` flag. The IOM sends this packet when its internal buffer becomes 50% full. The IOM continues to send the same packet every time it outputs one packet of data from the buffer until the host resumes the data stream.

9.1.4 Reader and writer threads

Reader and writer threads are created for each ACB and DMap. The threads perform conversion, movement and error correction of data.

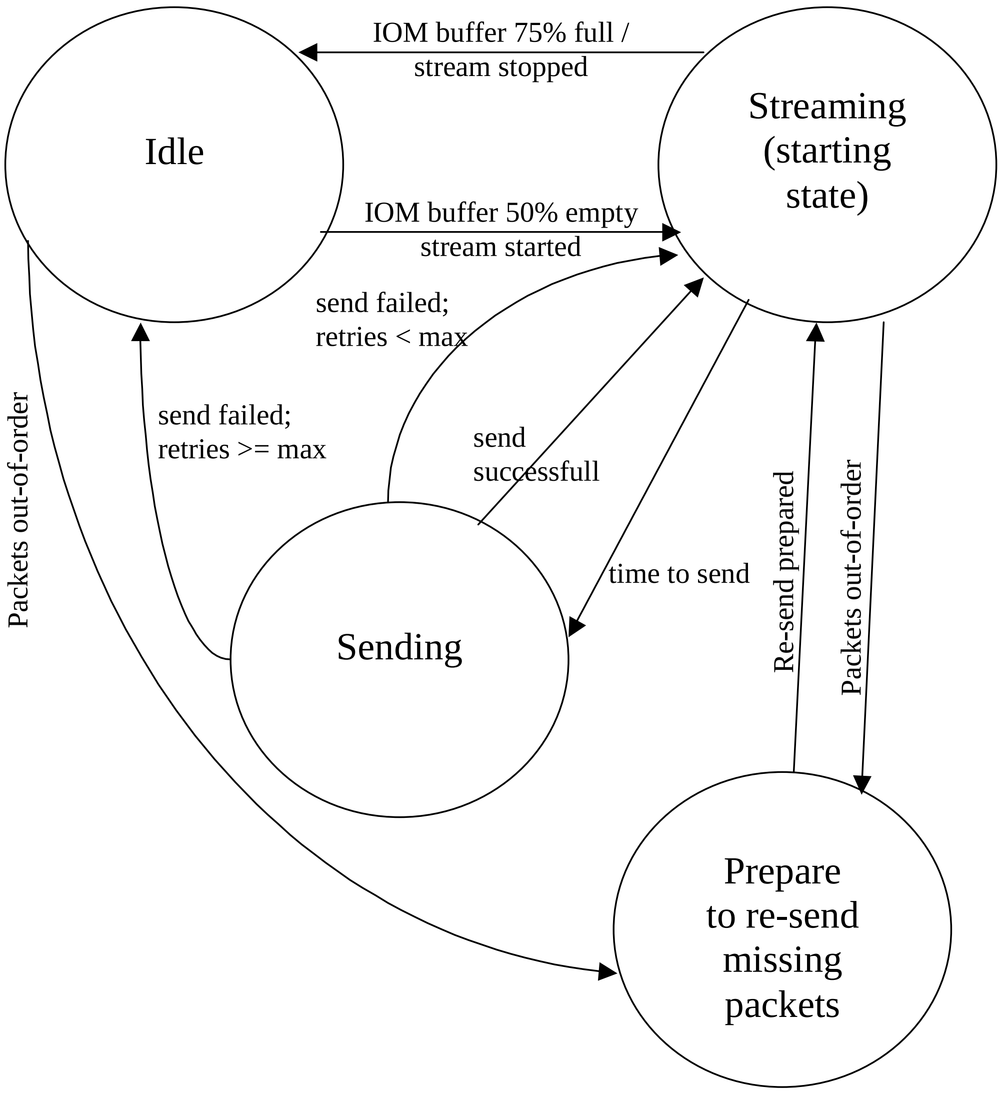
9.1.5 Error detection and error recovery

As we explained in 4.4.3, the DQE sends a request for a missing packet if it receives a packet in the packet sequence without receiving the previous one. The following diagram depicts state machine used to control input stream. Every packet in the stream is numbered from 1 to 65535 in `dqCounter` field. This way, the DQE knows if the sequence of data packet is out of order. In case a packet in the sequence is missing, DQE sends `DQCMD_RDFIFO` with the `dqCounter` equal to the number of the missing packet in the sequence. A packet can be requested a limited number of times.



Input streaming

In the case of output stream, the DQE state machine is a little more complicated because it includes states to control data flow from the DQE to the IOM. Every packet sent to the IOM is incrementally numbered. If the IOM finds that one of the packets in the input stream is missing, it sends a `DQCMD_WRFIFO` reply with the missing packet number. The DQE resends the requested data upon receiving this packet. The IOM limits the number of times it can request the packet. If the missing packet is not received at the time when it should be transferred into the layer buffer, the IOM firmware takes next available packet and releases the placeholder for the missing packet into free packet queue.

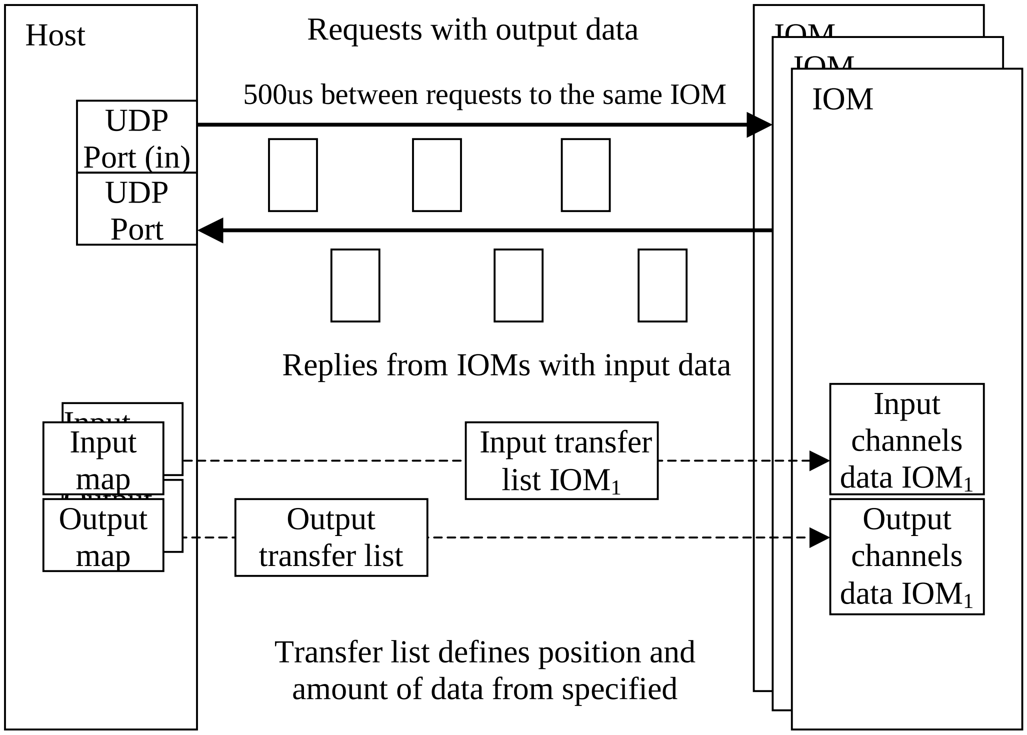

IOM buffer 75% full / stream stopped
Idle
Streaming (starting state)
IOM buffer 50% empty stream started
send failed; retries < max
send failed; retries >= max
send successfull
Packets out-of-order
time to send
Re-send prepared
Packets out-of-order
Sending
Prepare to re-send missing packets

Output streaming

For both the input and output stream, there is a potential situation when the received packet cannot be inserted into appropriate ring buffer. In this case, the receiving side replies to the transmitting side with the `DQERR_TOOOLD` error flag.

9.2 DMap structure and function

Direct data mapping is a mechanism that allows creating areas of input and output data that mirror data values on the input and output lines of networked IOMs. The following diagram depicts the structure of DMap operation.


Host
Requests with output data
500us between requests to the same IOM
UDP Port (in)
UDP Port
IOM
IOM
IOM
Replies from IOMs with input data
Input map
Output map
Input transfer list IOM$_1$
Output transfer list
Input channels data IOM$_1$
Output channels data IOM$_1$
Transfer list defines position and amount of data from specified

Every DMap has its input and output maps and can work with a single IOM. Two DMaps can work with the same IOM, however they have to address different layers.
The maximum size of a DMap is limited by the size to a maximum single packet size – 510 bytes. DMap allows representing data either in raw or engineering units (volts by default).
In DMap mode, IOM I/Os performs at the rate sufficient to update input points fast enough to provide a fresh input reading with every reply packet. Output runs at the rate to be able to update outputs before next portion of data arrives.
Setting up a DMap is a multi-stage operation, which is completely automated in the DQE. The user is not required to set up layer configuration, clocks and channel list. Instead, the DQE selects parameters that are most suited for the requested operation.

After the user specifies all channels he would like to see participating in data exchange, the DQE finalizes them. It parses the transfer list and sorts entries on a layer/subsystem basis. After that, the DQE sets up configuration, acquisition rate, and channel list for every layer involved. Finally, the DQE sets up the transfer list for the IOM involved. Every transfer list within a host has a unique ID (DMap ID.) The DQE calculates addresses of data points for each entry into the transfer list.

The IOM converts the transfer list into a table of addresses associated with the DMap ID. After the DQE starts the DMap operation, it takes data from the output DMap and sends `DQCMD_WRRD` packets on a periodic basis. When the IOM receives a `DQCMD_WRRD` packet, it start processing it based on the received DMap ID. The IOM takes transferred data sequentially and writes it according to the output physical addresses table created earlier. After that, it reads data from the input physical address table and stores it into the reply packet. The DQE mirrors the IOM operation and stores data into the input DMap.

Additionally, the DQE calls a conversion routine for each point of data to convert it from the raw representation into real-world values for both directions.

9.2.1 Direct memory mapping parameters

The user can overwrite automatically selected parameters for DMap operation by calling the appropriate function between finalization of the transfer list (`DqDmapInitOps()`) and enabling the DMap operation (`DqeEnable()`).

9.2.2 DaqBIOS commands used for communications with IOM

The DQE sequentially sends following commands to configure the IOM for DMap operation:

- `DQCMD_SETCFG`
- `DQCMD_SETCL`
- `DQCMD_SETCLK`
- `DQCMD_SETTRL`
- `DQCMD_FINISH`
- `DQCMD_SETMD`

9.2.3 Memory mapping for input and output data

The DQE detects whether the data is an input or output value based on the subsystem ID passed when the user adds channels to the DMap. All subsystems with even subsystem IDs are considered input subsystems. One DMap ID is assigned for both input and output DMaps. Both DMaps are allocated even if the size of input or output data is zero.

9.2.4 Error detection

Every `DQCMD_WRRD` packet sent from the host is sequentially numbered in `dqCounter` field of DaqBIOS protocol header. If the DQE receives a reply packet with non-sequential number, it sets up an `ePacketLost` flag and triggers the related event. There is no error correction mechanism supported.

9.2.5 Data replication over the network

DMap can be used for input data replication across local area network if workstations NICs are set into promiscuous mode and receive all reply packets from UDP interface.
DMap can also be used in homogenous networks of IOMs where IOMs exchange data between each other.

9.2.6 Associated events

As it was mentioned before, DMap supports only two events. Both events are set upon arrival of a reply packet and set by writer thread.
First one is `eDataAvailable` is set every time after writer thread transfers converted data into the input DMap.
The second one is `ePacketLost` when the DQE detects that one or more packets are missing.

9.3 Event notification

Event notification is implemented using OS signaling primitives. The DQE interface provides a set of functions to decouple PowerDNA application from underlying OS.
Event are controlled by three functions:

- int DqeSetEvent(pBCB pBcb, uint32 evtFlags);
  This function tells the DQE what events the user wants to associate with the BCB. The BCB can contain either DMap or ACB.
- int DqeGetEvent(pBCB pBcb, uint32* evtFlags);
  This function retrieves all event flags accumulated for this BCB.

- int DqeWaitForEvent(pBCB* ppBcb, int num, int WaitAll, int timeout, uint32* pEvents);
  This function waits for any event associated with any (or all) submitted BCBs to be asserted. Then it returns event flags for every BCB in the list.

9.4 Heartbeat

The heartbeat feature allows the DQE to keep track of which IOMs are available on the network. To achieve this goal, every IOM entry has special `ready` field. If this field is `TRUE`, IOM is alive.

9.4.1 Heartbeat and safe states

1. An IOM can have watchdog timer set up to reset the PowerDNA cube (hardware reset) if the idle thread doesn't clear this counter periodically. There are two ways to set it up:
   a. Use the command from the serial interface: "time watchdog N", where N is time in ms between counter resets. If this time is 0 watchdog timer is disabled
   b. Use `DQCMD_SETPRM` command with `IOMODE_NAMEDPRM` mode -- parameter name is `DQPRM_WATCHDOG`
2. An IOM can switch itself into shutdown mode or reset the PowerDNA cube from operation upon loss of communications. Host side is responsible to set this mode
   a. Use `DQCMD_SETPRM` command with `IOMODE_NAMEDPRM` mode -- parameter name is `DQPRM_COMMLOST`, to set up:
      i. Number of milliseconds before switching to shutdown mode
      ii. Reset flag to reset firmware and switch to initialization and then configuration mode upon reaching shutdown mode
      iii. Write 0 to switch this mode off

   When this mode is active, the IOM expects to receive any valid packet from the host side every N milliseconds. If there is no communication within this timeframe, the IOM sets up safe values on its outputs. The host has a special sticky entry (*Heartbeat Entry*, see below) with the `DQCMD_ECHO` command in the CQ scheduled to send earlier then communication timeout expires. The command will **not** always be sent to the IOM – the receiving thread will reset the *time-to-send* field of this entry each time a packet arrives from an IOM. This mechanism is especially useful for output streaming because the IOM doesn't send any replies if there are no errors in the stream.

9.4.2 Heartbeat and Moving Token Mechanism

In each command queue, there can be a special entry – Heartbeat Entry. It is given a special treatment by both receiving and sending thread.

9.4.3 Heartbeat Processing in Receiving Thread

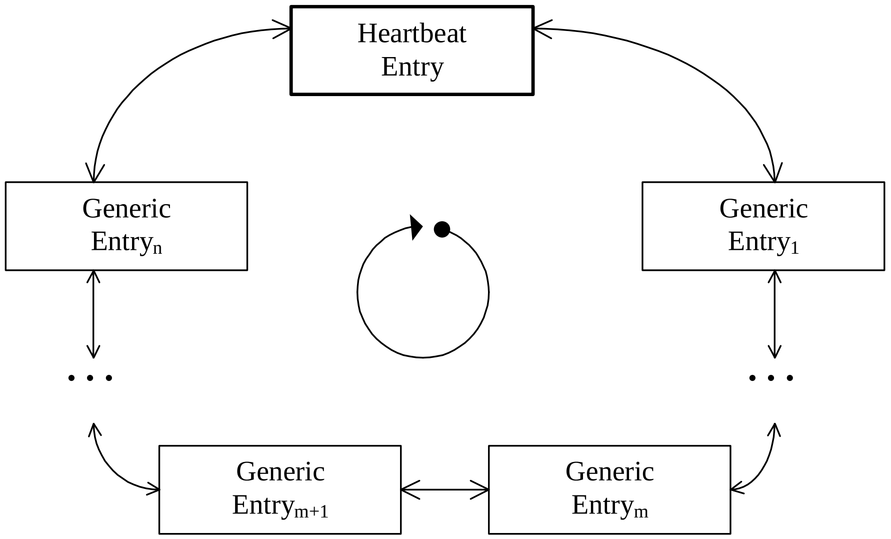
Each time a receiving thread gets a packet from an IOM, it resets the *time-to-send* and *time-before-timeout* fields in the Heartbeat Entry to the maximum and clears *waiting-for-answer* flag. It also marks the IOM "available" in the IOM table. This will ensure that the `DQCMD_ECHO` command, associated with the Heartbeat Entry, will be sent out to the IOM only in case of prolonged silence, and even if it was sent but IOM responded in the mean time, there will be no alarm.

9.4.4 Heartbeat Processing in Sending Thread

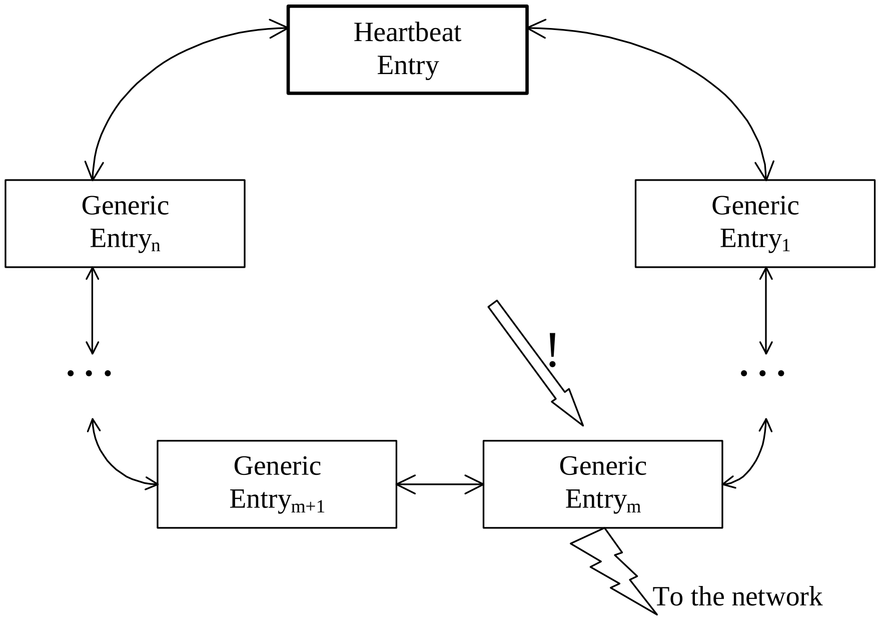
A sending thread always starts processing the Command Queue (CQ) from the Heartbeat Entry. If it ever reduces *time-before-timeout* to zero, the IOM will be marked "unavailable" and no other commands except the Heartbeat Entry's DQCMD_ECHO will be sent.
In normal operation, Heartbeat Entry will also serve as a **token**, marking the place where we last stopped processing the CQ. After going through the whole CQ, if the sending thread has sent any command this cycle, the sending thread moves Heartbeat Entry from its current position directly after the corresponding entry (the one associated with the command just sent). This ensures the rotation between all commands in the CQ.

Here is how the mechanism described above works:

Sending thread starts going through the CQ from Heartbeat Entry:



Sending thread finds a ready-to-send entry and sends the packet:



9.5 Programming in synchronous mode

Programming in synchronous mode can be done with or without invoking the DQE. Most DaqBIOS commands include a request sent from host and a reply sent back from the IOM to the requesting host.

When the library is used without the DQE, each function call sends a packet to the IOM and waits for the reply from it. Thus, the function does not return control to the calling process before either the IOM sends a reply or the timeout expires. All underlying function calls occurs in the context of the user application process. The timeout in milliseconds is set in `DqOpenIOM()` or `DqSetTimeout()`. There is a separate timeout for each IOM.

When the DQE is running, a function call does not send a packet directly but adds the packet into the command queue (CQ.) This entry receives a unique system-wide request ID. Then the function call relinquishes control to the OS on the wait function. It waits until either receiving an event or a timeout.
On the next clock cycle, the sending thread finds the new entry in the command queue and sends the packet to the destined IOM. At the same time, it decreases the retry counter in the command queue entry.
When the receiving thread receives the packet with the same request ID, it deletes the command queue entry and copies the packet data into the buffer belonging to the waiting function call.
Every time the sending thread receives a clock cycle, it decreases the timeout value in the command queue entry. When a timeout occurs, it sends the same packet again and decreases the retry counter. This way, every synchronous command is repeated several times before the sending thread releases the waiting user call with a timeout.
If the IOM doesn’t reply, then the total timeout will equal the reply timeout multiplied by the number of retries multiplied by the DQE clock period.
`DqSetTimeout()` can change the wait time on the fly for all IOMs addressed by the DQE. The user cannot change the timeout time for each IOM.

9.6 Using DaqBIOS Engine (DQE)

9.6.1 Setting up the DQE parameters

When the user creates the DQE using `DqStartDQEngine()`, several parameters can be adjusted defining behavior of the DaqBIOS Engine.

8 Default parameters

When user creates the DQE, it calls `DqStartDQEngine()`. The user can supply two parameters to the DQE instance before its creation. One parameter is `period_us` which defines how often the sending thread can wake up (in microseconds) or the base clock cycle. Different OSes have different minimal resolutions. For example, Windows and Linux have minimum resolution of 1ms (1000µs).

The other parameter is `pDQEPRM` - a pointer to `DQEPRM`. If this parameter is `NULL`, the DaqBIOS Engine is initialized with default parameters. If a user wants to adjust just a few parameters upon DQE creation, he can supply pointers to the parameters he wants to change and set all other pointers in the `pDQEPRM` to `NULL`.

All major and user adjustable DQE parameters are encapsulated in the following structure:

```
// DQE parameters
typedef struct {
    uint32* timeout;             // reply wait timeout
    uint32* retries_async;       // number of retries for asynchronous commands
before
                                 // return an error
    uint32* retries_receive;     // number of retries while receiving stream
before
                                 // placing filler
    uint32* retries_send;        // number of retries while sending stream upon
dumping
                                 // packet
    uint32* max_inbound_packet;  // maximum packet size from IOM
    uint32* max_outbound_packet; // maximum packet size send to IOM
    uint32* abort_after;         // when streaming abort operation after number of
                                 // lost packets
    uint32* use_protocol;        // switch between DQ_TS and DQ_VT (RESERVED)
    uint32* packets_at_once;     // maximum number of packets to one IOM upon one
tick
} DQEPRM, *pDQEPRM;
```

By default these parameters are initialized with following values.

```
#define DQE_TIMEOUT              10     // ticks of DQE
#define DQE_RETRIES_ASYNC        3      // retries
#define DQE_RETRIES_RECEIVE      3      // retries
#define DQE_RETRIES_SEND         3      // retries
#define DQE_INBOUND_PACKETSZ     512    // packet size
#define DQE_OUTBOUND_PACKETSZ    512    // packet size
#define DQE_ABORT_AFTER          10     // retries
#define DQE_USE_PROTOCOL         DQ_TS  // reserved
#define DQE_PACKETS_AT_ONCE      3      // retries
```

9 What parameters can be changed and What is their effect

Following parameters can be adjusted by user and affect DQE behavior:

`timeout` defines number of sending thread clock cycles before packet associated with command queue entry is either discarded or resent.

`retries_async` defines number of attempts performed by the DQE to send packet to the IOM prior to declaring timeout.

`abort_after` defines number of sequentially lost packets before DQE aborts streaming operation and returns an error.

`packets_at_once` defines how many packets can be sent to the same IOM in a single sending thread clock cycle. In the ACB streaming operation, this parameter limits number of requests sent to the IOM in one clock period. The IOM processor limits the number of packets it can process thus if the IOM cannot process a packet on time it will be dropped.

Other parameters are reserved for future use.

9.6.2 Program structure in ACB mode

ACB functions were written to simplify the required structure of the user program to achieve a stream of data.
First, the DQE engine has to be started and all IOMs involved opened. Upon opening of IOM, the DQE sends the `DQCMD_ECHO` command to each of them and then follows with `DQCMD_GETCAPS` command to retrieve information about every layer installed with the `DqAcbInitOps()` call.

10 Buffer parameters

When the user creates an ACB, he specifies what IOM layer and subsystem (of that layer) is involved in the operation.
After that, the user calls `DqAcbInitOps()` to initialize parameters of the stream. Besides acquisition parameters and layer configuration, the user specifies the ACB parameters in `pACBCFG pAcbCfg`.

`ACBCFG` is a complex structure containing multiple parameters defining an ACB. Not all of them can be set and adjusted by user.

```
// ACB description
// Contains information
typedef struct _pACBCFG {
    uint32 samplesz;   // raw sample size, bytes
    uint32 valuesz;    // converted value size, bytes
    uint32 scansz;     // scan size, samples/values
    uint32 framesize;  // number of scans in the frame, max
    uint32 frames;     // frames in the buffer
    uint32 ppevent;    // packets per ePacketDone event
    uint32 mode;       // mode of operations: Single, Cycle, Recycled, error
handling
    uint32 dirflags;   // transfer direction and additional flags
    uint32 maxpktsize; // how much data to accumulate in the packet before sending
(0=default)
    uint32 hwbufsize;  // how much data to keep on the PowerDNA cube (0 = default)
    uint32 hostringsz; // number of packets in the host ring buffer (0 = default)
    uint32 wtrmark;    // percent of the ring buffer queue packets kept in case
IOM reports an error
    double eucoeff;    // engineering unit coefficient to multiply voltage data by
    double euoffset;   // engineering unit offset
} ACBCFG, *pACBCFG;
```

`samplesz` – the DQE calculates samples size in bytes and stores it in the user-supplied structure. Sample size depends on layer type.

`valuesz` – the DQE calculates the size of each point of data in ACB.

`scansz` – the DQE calculates scan size based on supplied channel list size and configuration (e.g. is timestamp enabled and its size.)

`framesize` – the user supplies size of frame, in samples. If the user supplies zero as a frame size, the DQE selects the frame size equal to the amount of data delivered in one packet of maximum size.

`frames` – the user supplies the number of frames in the buffer. If the user sets it to zero, the DQE selects two frames per buffer as a default setting.

`ppevent` – How often the user wants to receive `ePacketDone` events. An `ePacketDone` event is generated upon receiving and processing selected number of frames. Default value is one.

`mode` – defines what type of buffer mode to use: `DQ_ACBMODE_SINGLE`, `DQ_ACBMODE_CYCLE` or `DQ_ACBMODE_RECYCLED`. Default mode is `DQ_ACBMODE_CYCLE`.

`dirflags` -- defines the direction of operation and data conversion settings. The following flags are defined:

```
// Defines for data conversion function (dirflags)
#define DQ_ACB_DIRECTION_INPUT     0x0     // dir-in
#define DQ_ACB_DIRECTION_OUTPUT    0x1     // dir-out
#define DQ_ACB_DATA_SINGLE         0x100   // data in the ACB is <float>
#define DQ_ACB_DATA_DOUBLE         0x200   // data in the ACB is <double>
#define DQ_ACB_DATA_RAW            0x300   // data in the ACB is Raw
#define DQ_ACB_DATA_EUNITS         0x400   // double in engineering units
#define DQ_ACB_DATA_TSCOPY         0x1000  // copy timestamp into ACB is
available
#define DQ_ACB_NOCALIBRATION       0x2000  // do not perform software
calibration
```

`maxpktsize` – limits the maximum packet size. A smaller packet size improves response time but decreases performance. We recommend using default value.

`hwbufsize` – sets up the size of the hardware buffer in packets. We recommend using default value.

`hostringsz` – number of packets in the packet ring buffer allocated by the DQE. We recommend using default value.

`wtrmark` – Defines of how much data to keep in the output packet ring buffer in case the IOM requests missing packets. We recommend using default value of 50%.

Another parameter `ScanBlock` controls how many scans the IOM has to put into one packet. By default, the IOM tries to pack as many scans as possible into one packet. This decision decreases number of packets and improves throughput.

11 Available events

Most of the time the user is interested in the following events:

```
eFrameDone
eBufferDone
eDataAvailable
```

and errors:

```
eBufferError
ePacketLost
ePacketOOB
```

12 User application design in pseudo-code

The following pseudo-code samples depict typical user application using ACB and DMap techniques. All user-supplied functions in the pseudo-code start with `dquser_`.

9.6.2.1.1 Program structure with input ACB and event handler (input direction)

1. Start DQE engine

DqStartDQEngine()

2. Create and initialize host and IOM sides

```
        DqAcbCreate()
        DqAcbInitOps()
        DqeSetEvents()
```

3. Start operation

```
        DqeEnable(TRUE, …)
```

4. Process data

```
        while(running) {
                DqeWaitForEvent()
                if (events_to_process)
                        DqAcbGetScans()
                        dquser_process_received_scans()
                else if (error_conditions)
                        dquser_process_error_conditions()
        }
```

5. Stop operation

```
        DqeEnable(FALSE, …)
```

6. Clean up

```
        DqAcbDestroy()
        DqStopDQEngine()
```

The user can rely on either the `DqAcbGetScans()` or the `DqAcbGetScansCopy()` function to retrieve scans from an ACB. `DqAcbGetScans()` returns a pointer to the date residing in the ACB and amount of data user can copy.
`DqAcbGetScansCopy()` copies requested data into the user buffer.
`DqAcbGetScansCopy()` is a safer function because it copies data into the user buffer and only then releases that part of an ACB buffer for a new data.
`DqAcbGetScans()` works faster because it doesn't copy data from one place in the memory to another. However, the DQE releases the part of an ACB containing returned samples right away. In other words if user application is not fast enough, the user will lose data.

9.6.2.1.2 Program structure with output ACB and event handler (output direction)

```
1. Start DQE engine
        DqStartDQEngine()

2. Create and initialize host and IOM sides
        DqAcbCreate()
        DqAcbInitOps()
        DqeSetEvents()

3. Start operation
        DqeEnable(TRUE, …)

4. Process data
        dquser_fill_buffer_with_output_data()
        while(running) {
                DqeWaitForEvent()
                if (events_to_process)
                        DqAcbPutScans()
                        dquser_add_scans_to_the_buffer()
                else if (error_conditions)
                        dquser_process_error_conditions()
        }

5. Stop operation
        DqeEnable(FALSE, …)

6. Clean up
        DqAcbDestroy()
        DqStopDQEngine()
```

There are two ways to put new samples into the buffer.
First one uses `DqAcbPutScansCopy()`. The user buffer with output data should be prepared. The function copies data into an ACB and then return the number of samples it was actually able to copy.
The second one make use of `DqAcbPutScan()` function. This function is called first to determine how many scans the buffer can accommodate and the

starting address. After that, the user prepares data and copies it directly into an ACB.

9.6.2.1.3 Program structure with output DMap and event handler (both directions)

```
1. Start DQE engine
        DqStartDQEngine()

2. Create and initialize host and IOM sides
        DqDMapCreate()

3. Add channels into DMap
        DqDmapSetEntry()
        …
        DqDmapSetEntry()

        DqDmapInitOps()
        DqeSetEvents()

4. Start operation
        DqeEnable(TRUE, …)

5. Process data
        dquser_fill_ output_values()
        while(running) {
                DqeWaitForEvent()
                if (DQ_eDataAvailable)
                        dquser_read_input_write_output()
                else if (DQ_ePacketLost)
                        dquser_process_error_conditions()
        }

6. Stop operation
        DqeEnable(FALSE, …)

7. Clean up
    DqDmapDestroy()
    DqStopDQEngine()
```

9.6.2.1.4 Program structure with periodic timer

There is no significant difference between event-driven and timer-driven application.

An ACB-based, periodic-timer based application appears as follows:

1. The user configures and starts ACB operation
2. The user signs up for a timer notification with the OS
3. On a timer event, the user retrieves the number of scans available for input by calling `DqAcbGetScans()` with minimum request size argument large enough for processing
4. If there is enough scans, the user processes them and/or outputs them onto the screen
5. The user stops operation and closes application the shown way.

A DMap-based, periodic-timer based application is even simpler:

1. The user configures and starts DMap
2. Upon timer tick, the user application reads input data from the input memory map and writes output data into output memory map
3. The user stops DMap operation when completed

9.6.3 Potential DQE problems, performance tuning and troubleshooting

The maximum performance of the PowerDNA system is limited by the:

1. Ability of the IOM to transfer data from the layer memory into the CPU memory
2. Ability if the IOM to process incoming and outgoing packets
3. Network bandwidth
4. Ability of the host to send and receive packets

One of the ways to tweak performance on the PowerDNA cube side is to adjust number of packet buffers allocated for I/O data. This can be done by calling `DqCmdSetParameters()` with `DQ_IOPRM_NBUFS` as a named parameter. Every buffer occupies 576 bytes of the memory. The default value is 1024 packet buffers.

At the edge of performance, an IOM can work unreliably. The reason is that when an IOM works at full utilization of embedded processor resources and an error happens, the processor can have no time to process it. Subsequently, the embedded processor will lose another packet and cause another packet request from the host.

Thus, to utilize an IOM at the maximum possible throughput, the user should sacrifice error correction for performance. This can be done using DQE parameter flag.

10 Appendix

10.1 Configuring a Second Ethernet Card Under Windows XP

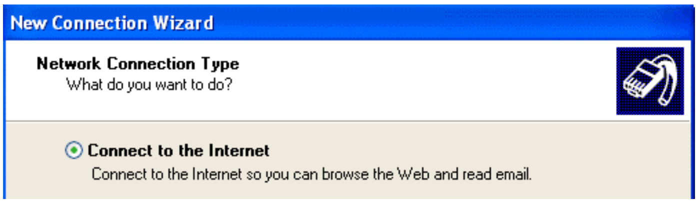
10.1.1 Set Up Your Ethernet Card (NIC)

- If you already have an Ethernet card installed, skip ahead to the next section, *Configure TCP/IP*.
- If you have just added an Ethernet card, to install it:

1. From the *Start* menu, select *Control Panel*, and click *Printers and Other Hardware*.
2. From the menu on the left, click *Add Hardware* and follow the on-screen instructions.
   We recommend that you allow Windows XP to search for and install your Ethernet card automatically. If Windows XP does not find your Ethernet card, you will need to install it manually by following the manufacturer's instructions.
3. Once your Ethernet card has been installed, continue to the next section.

10.1.2 Configure TCP/IP

1. From the *Start* menu, select *Control Panel*.
2. Under the heading *Pick a category*, click *Network and Internet Connections*.
3. Under *pick a Control Panel icon*, click *Network Connections*.
   a. If you see an icon under *LAN or High-Speed Internet* heading for your second NIC, skip ahead to step 10.
   b. If there is no icon under *LAN or High-Speed Internet* for your second NIC, proceed to step 4.
4. From the menu on the left, click *Create a new connection* to launch the *New Connection Wizard*.
5. Click *Next* and proceed to the *Network Connection Type* window.
6. Select *Connect to the Internet* and click *Next*.

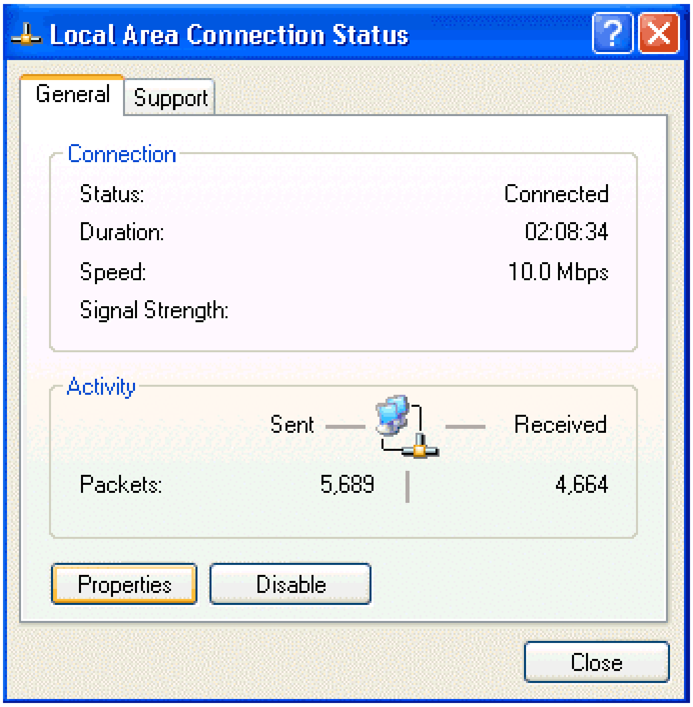

New Connection Wizard

Network Connection Type
What do you want to do?

● Connect to the Internet
Connect to the Internet so you can browse the Web and read email.

7. Select *Set up my connection manually* and click *Next*.
8. Select *Connect using a broadband connection that is always on* and click *Next*.
9. Click *Finish*.

10. In the *Network Connections* window, double-click the second icon under *LAN or High-Speed Internet*. In the next window (see illustration below), click *Properties*.



Connection

Status: Connected
Duration: 02:08:34
Speed: 10.0 Mbps
Signal Strength:

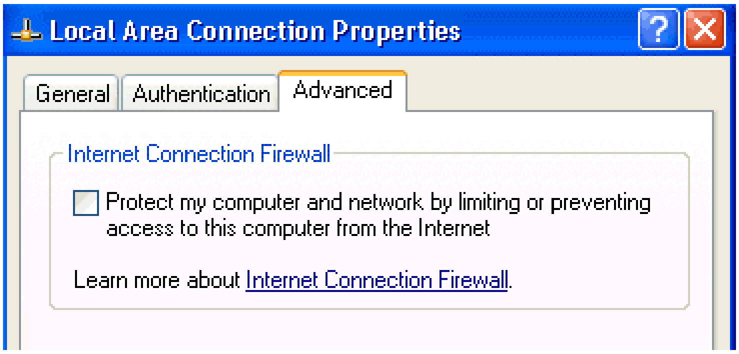
Packets: 5,689 4,664

11. Click the *General* tab, click once on *Internet Protocol (TCP/IP),* then click *Properties*.
12. Click the *General* tab, click *Use the following IP addresses***,** and in the corresponding boxes, enter `192.168.100.1` for the *IP address*, `255.255.255.0` for the *Subnet Mask* and leave blank the *router (or default gateway)* information.
13. Click *Use the following DNS server addresses*.
14. Make sure the *Preferred DNS server* box and the *Alternate DNS server* box are blank.
15. Click *OK* or *Close* until you return to the *Network Connections* window.
16. Close the *Network Connections* window.

10.1.3 Troubleshooting

If you encounter problems connecting to the network, first check to make sure the Windows XP Internet Connection Firewall is turned off. Follow the instructions below:

1. From the *Start* menu, select *Control Panel*.
2. Under the heading *Pick a category*, click *Network and Internet Connections*.
3. Under *pick a Control Panel icon*, click *Network Connections*.
4. Double-click the icon under *LAN or High-Speed Internet*. In the next window, click *Properties*.
5. Click the *Advanced* tab and **uncheck** the box *Protect my computer and network by limiting or preventing access to this computer from the Internet* (see illustration below).


Local Area Connection Properties

General | Authentication | Advanced

Internet Connection Firewall

☐ Protect my computer and network by limiting or preventing access to this computer from the Internet

Learn more about Internet Connection Firewall.

6. Click *OK* or *Close* until you return to the *Network Connections* window.
7. Close the *Network Connections* window.

10.1.4 Using the Windows XP Alternate Configuration Setting

If you're using a computer with only one Ethernet port, such as a laptop, you can configure Windows XP to automatically switch settings depending on which network it's connected.

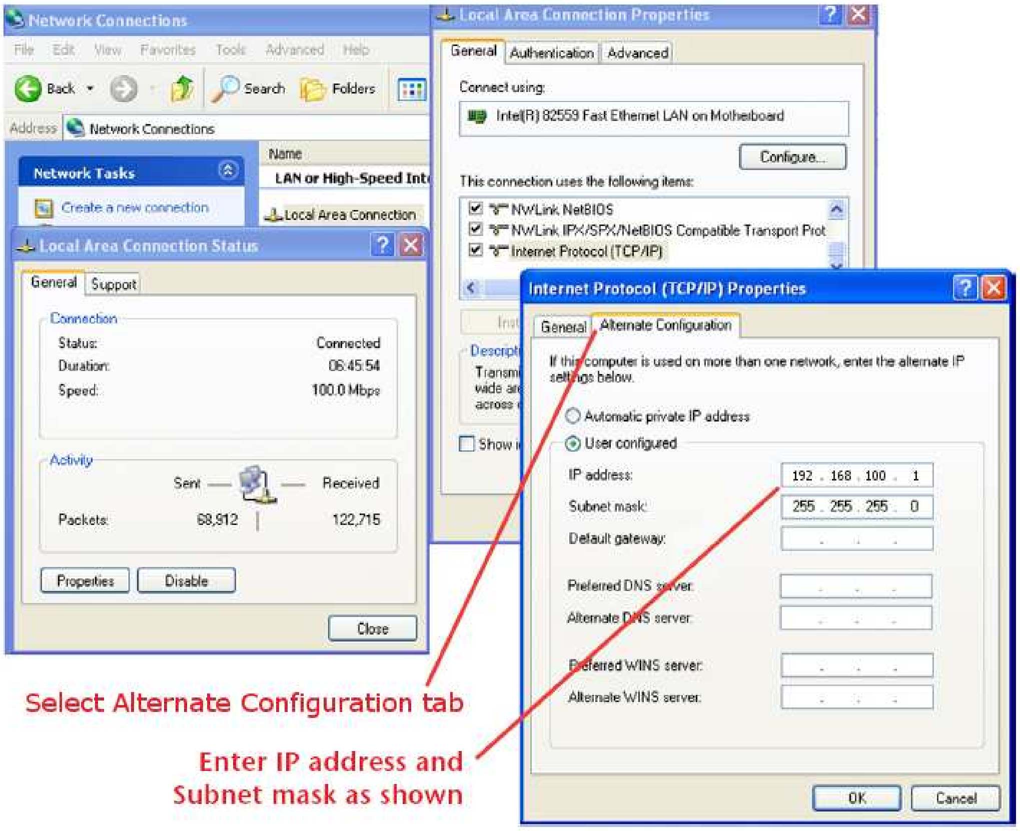
Windows XP users have the ability to configure a second IP address setting under the control panel that will allow Windows to pick the correct computer IP setting based on the device that it finds connected to the Ethernet port. Under this configuration your primary IP setting is configured for *Obtain IP Address Automatically* for connection to your company Network, and your secondary IP setting (*Alternate Configuration*) is configured for 192.168.100.1 with a subnet mask of 255.255.255.0 for connection to the PowerDNA cube.

The following steps allow you to configure your alternate IP address, starting at the Control Panel.

1. Double click on *Network Connections*
2. Double click on *Local Area Connections*
3. Click on the Properties button
4. Select *Internet Protocol (TCP/IP)* and click on the *Properties* button
5. Select the *Alternate Configuration* tab
6. Select *User Configured*
7. Enter `192.168.100.1` for the *IP address*
8. Enter `255.255.255.0` for the *Subnet mask*

9. Close all open configuration windows using *OK* or *Close*

Use the following screen shot to properly configure the *Alternate Configuration* tab located under the Windows XP network configuration screen located in the Windows XP control panel.


Network Connections
File Edit View Favorites Tools Advanced Help
Back Search Folders
Address Network Connections
Network Tasks
Create a new connection
Name
LAN or High-Speed Int
Local Area Connection
Local Area Connection Status
General Support
Properties Disable Close
Local Area Connection Properties
General Authentication Advanced
Configure...
Internet Protocol (TCP/IP) Properties
General Alternate Configuration
OK Cancel
Select Alternate Configuration tab
Enter IP address and Subnet mask as shown

Connection
Status: Connected
Duration: 06:45:54
Speed: 100.0 Mbps

Activity
Sent — Received
Packets: 68,912 | 122,715

Connect using:
Intel(R) 82559 Fast Ethernet LAN on Motherboard

This connection uses the following items:
☑ NWLink NetBIOS
☑ NWLink IPX/SPX/NetBIOS Compatible Transport Prot
☑ Internet Protocol (TCP/IP)

If this computer is used on more than one network, enter the alternate IP settings below.

○ Automatic private IP address
● User configured

IP address:	192 . 168 . 100 . 1
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .
Preferred DNS server:	. . .
Alternate DNS server:	. . .
Preferred WINS server:	. . .
Alternate WINS server:	. . .

Once you have this configuration in place your computer will look for the attached device on your Ethernet port during “Boot Up” or during a Windows “Log On” operation. If it sees a powered on PowerDNA cube connected to the Ethernet port, it will automatically switch to using the secondary IP address. If the computer sees a DHCP network connected to the Ethernet port, it will use the primary IP configuration and negotiate an IP address with your company network as required.

- If you are in the office and you want to check your email: Plug in the Ethernet cable for your company’s network connection into your computer and either power up your computer and log onto the network as you normally do, or if your computer is already powered on, perform a Windows “Log Off” and then a “Log On” and log onto your company network as you normally do.
- If you are working in the field with a PowerDNA cube: Plug in the Ethernet cable from the data acquisition system into your computer and make sure that the data acquisition system is powered on. Then either power up your computer and bypass your network log

on screens, or if your computer is already powered on, perform a “Log Off” and then a “Log On” and bypass your network logon screens.

10.2 Configuring a Second Ethernet Card Under Windows 2000

10.2.1 Set Up Your Ethernet Card (NIC)

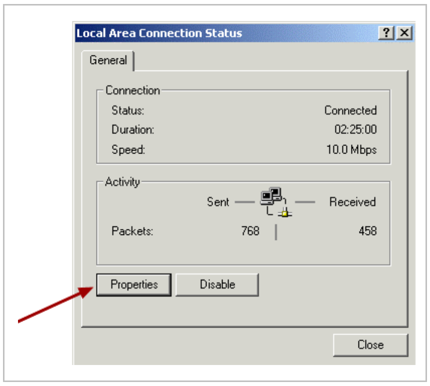
Windows 2000 will normally detect and install your Ethernet card and TCP/IP automatically. To check that your card has been installed, run through the following steps.

From the *Start* menu, select *Settings* and then select *Network and Dial-up Connections*.

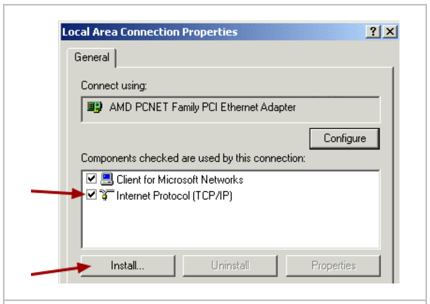
1. . If you see a *Local Area Connection* icon, your Ethernet card has been detected and installed, skip ahead to the section *Configure TCP/IP*. If you do not see this icon, proceed to step 3.
2. From the *Start* button, select *Settings*, then *Control Panel*. Double-click on the *Add/Remove Hardware* icon and follow the on-screen instructions. We recommend that you allow Windows 2000 to search for and install your Ethernet card automatically. If Windows 2000 does not find your Ethernet card, you will need to install it manually by following the manufacturer's instructions.
3. Once your Ethernet card has been installed, click *OK* and continue with the next section.

10.2.2 Installing TCP/IP

1. From the *Start* menu, select *Settings* and then select *Network and Dial-up Connections*.
2. In the *Network and Dial-up Connections* window, double-click on the *Local Area Connection 2* icon.
3. In the *Local Area Connection 2 Status* window, click *Properties*:



Status: Connected
Duration: 02:25:00
Speed: 10.0 Mbps
Packets: 768 458


Local Area Connection Properties

General

Connect using:

AMD PCNET Family PCI Ethernet Adapter

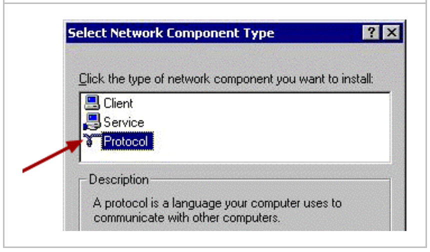
Configure

Components checked are used by this connection:

☑ Client for Microsoft Networks
☑ Internet Protocol (TCP/IP)

Install... Uninstall Properties

4. If *Internet Protocol (TCP/IP)* is listed, make sure the box next to it contains a check mark, and go to *Configure TCP/IP*.


Select Network Component Type

Click the type of network component you want to install:

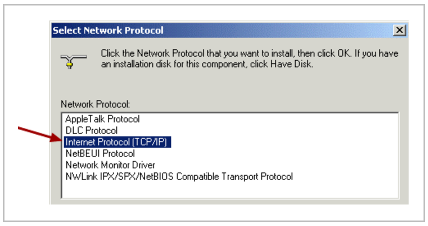
Client
Service
Protocol

Description

A protocol is a language your computer uses to communicate with other computers.

5. If *Internet Protocol (TCP/IP)* is not listed, click on *Install*.

6. In the next window, double click on *Protocol*.


Select Network Protocol

Click the Network Protocol that you want to install, then click OK. If you have an installation disk for this component, click Have Disk.

Network Protocol:

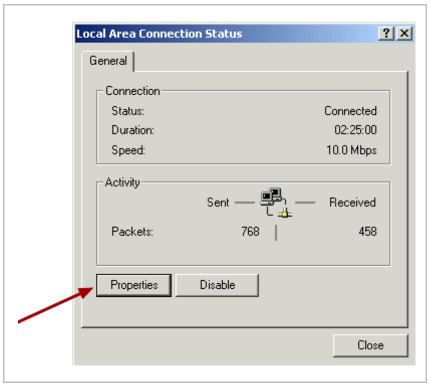
AppleTalk Protocol
DLC Protocol
Internet Protocol (TCP/IP)
NetBEUI Protocol
Network Monitor Driver
NWLink IPX/SPX/NetBIOS Compatible Transport Protocol

7. Select *Internet Protocol (TCP/IP)*, and click *OK*

8. Make sure the box beside *Internet Protocol (TCP/IP)* contains a check mark, and proceed to the next section, *Configure TCP/IP*.

10.2.3 Configure TCP/IP

1. From the *Start* menu, select *Settings* and then select *Network and Dial-up Connections*.
2. In the *Network and Dial-up Connections* window, double-click on the *Local Area Connection 2* icon.


Local Area Connection Status

General

Connection

Status:	Connected
Duration:	02:25:00
Speed:	10.0 Mbps

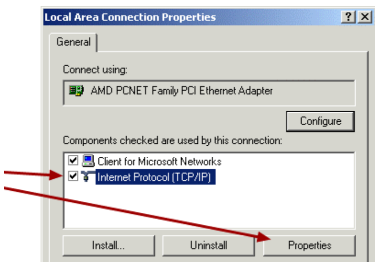
Activity

	Sent	Received
Packets:	768	458

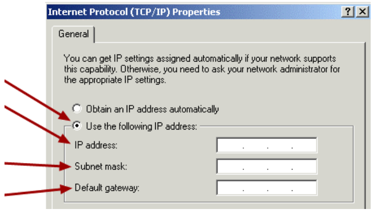
Properties Disable

Close

3. In the *Local Area Connection 2 Status* window, click *Properties*:


Local Area Connection Properties
General
Connect using:
AMD PCNET Family PCI Ethernet Adapter
Configure
Components checked are used by this connection:
Client for Microsoft Networks
Internet Protocol (TCP/IP)
Install...
Uninstall
Properties

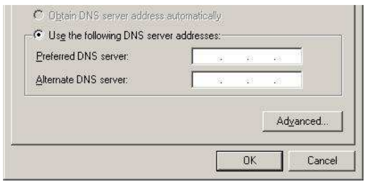
4. Click once on *Internet Protocol (TCP/IP).* Then click *Properties.*


Internet Protocol (TCP/IP) Properties
General
You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.
Obtain an IP address automatically
Use the following IP address:
IP address:
Subnet mask:
Default gateway:

5. Select *Use the following IP address*, and type `192.168.100.1`

   In the *Subnet mask* box type `255.255.255.0`.

   Leave the *Default Gateway* box blank.


Obtain DNS server address automatically
Use the following DNS server addresses:
Preferred DNS server:
Alternate DNS server:
Advanced...
OK
Cancel

6. Select *Use the following DNS server addresses* and:
   Make sure the *Preferred DNS server* box and the *Alternate DNS server* box are blank.

7. Click *OK*, click *OK* in the *TCP/IP Properties* window, click *OK* in the *Local Area Connection* window and click *Close* in the *Local Area Status* window.

8. Close the *Network and Dial-up Connections* window.

10.3 Configuring a Second Ethernet Card Under Windows NT

10.3.1 Set Up Your Ethernet Card (NIC)

If you installed your Ethernet interface ***before*** *(or at the same time as) you installed Windows NT*, then the system should have automatically detected it and you should proceed to the next section, Install and Configure TCP/IP. Optionally, you may follow steps 1-3 below to confirm that your interface is recognized.

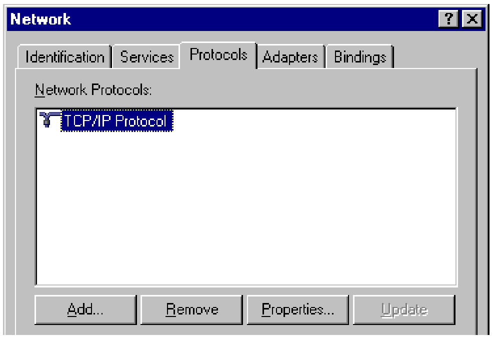
If you obtained an Ethernet interface ***after*** *Windows NT was already on your computer*, then do the following:

1. From the *Start* menu, select *Settings* and then select *Control Panel*.
2. Double-click on the *Network* icon.

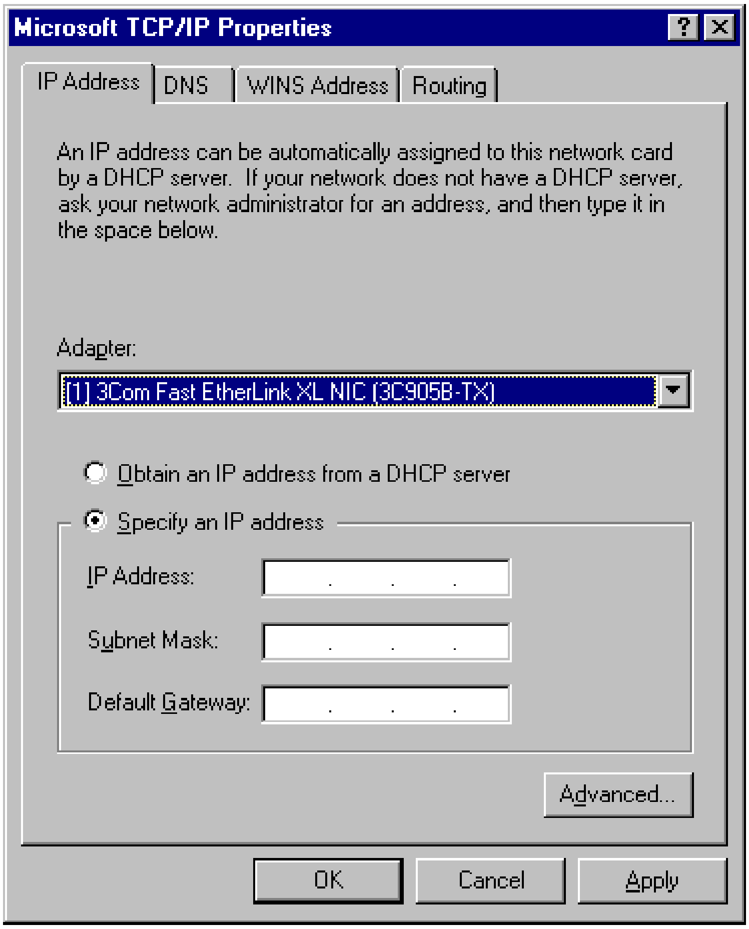
3. Click on the tab labeled *Adapters*. You should then see an entry for your Ethernet card. If you do not see one, continue to step 4 to install it. Otherwise, click *OK* and skip ahead to *Install and Configure TCP/IP*.
4. Click *Add...* and follow the on-screen instructions. Select your Ethernet card from the list shown, or, if it is not included in the list, click *Have Disk...* and insert the diskette that came with the card. Even if your card does appear in the list, it's a good idea to use the diskette to make sure you have the latest drivers.
5. Restart your computer if Windows gives you the option to do so. Wait for the system to restart before continuing with the next section.

10.3.2 Install and Configure TCP/IP

1. From the *Start* menu, select *Settings* and then *Control Panel*.
2. Double-click on the *Network* icon, then click the *Protocols* tab.
3. In the list of *Network Protocols*, look for *TCP/IP Protocol*. If you don't see it, click *Add...*, select *TCP/IP Protocol*, and then click *OK*.
4. Select *TCP/IP Protocol* in the list of *Network Protocols* and then click *Properties...* A *Microsoft TCP/IP Properties* window will open.



5. Click on the *IP Address* tab if it is not already selected.
   a. Make sure that the radio button next to *Specify an IP address* is selected.
   b. Enter `192.168.100.1` for *IP Address*, `255.255.255.0` for *Subnet Mask*, and leave blank the *Gateway Address* (in the *Default Gateway* box.)



Microsoft TCP/IP Properties

An IP address can be automatically assigned to this network card by a DHCP server. If your network does not have a DHCP server, ask your network administrator for an address, and then type it in the space below.

Adapter: [1] 3Com Fast EtherLink XL NIC (3C905B-TX)

○ Obtain an IP address from a DHCP server

● Specify an IP address

IP Address: . . .

Subnet Mask: . . .

Default Gateway: . . .

6. Click on the *DNS* tab.
   - Leave blank the *Host Name* and *Domain* fields.
7. Click *OK* to close the *Microsoft TCP/IP Properties* window.
8. Click *Close* to close the *Network* control panel.
9. Restart your computer.
10. You should now be able to access network-based services.

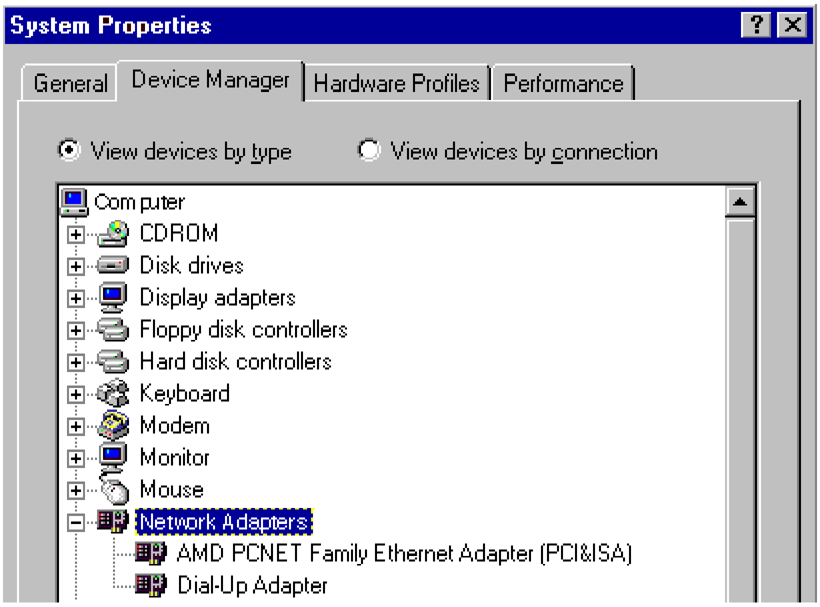
10.4 Configuring a Second Ethernet Card Under Windows 95/98/SE/ME

10.4.1 Set Up Your Ethernet Card (NIC)

If you installed your Ethernet card ***before*** *(or at the same time as) you installed Windows 95/98/Me*, then the system should have automatically detected it and you should proceed to the next section, Install TCP/IP. Optionally, you may follow steps 1-3 below to confirm that your card is recognized.

If you obtained an Ethernet interface ***after** Windows 95/98/Me was already on your computer*, then do the following:

1. From the *Start* menu, select *Settings* and then select *Control Panel*.
2. Double-click on the *System* icon, then click on the tab labeled *Device Manager*.
3. Double-click on *Network adapters* to display a list of the network interfaces that are installed on your computer. If you see two entries other than the *Dial-Up Adapter*, one is your second *Ethernet card*. Skip ahead to *Install TCP/IP*. If you do *not* see your second *Ethernet card*, continue to step 4 to install it.



(● View devices by type) ○ View devices by connection

Computer
- CDROM
- Disk drives
- Display adapters
- Floppy disk controllers
- Hard disk controllers
- Keyboard
- Modem
- Monitor
- Mouse
- Network Adapters
  - AMD PCNET Family Ethernet Adapter (PCI&ISA)
  - Dial-Up Adapter

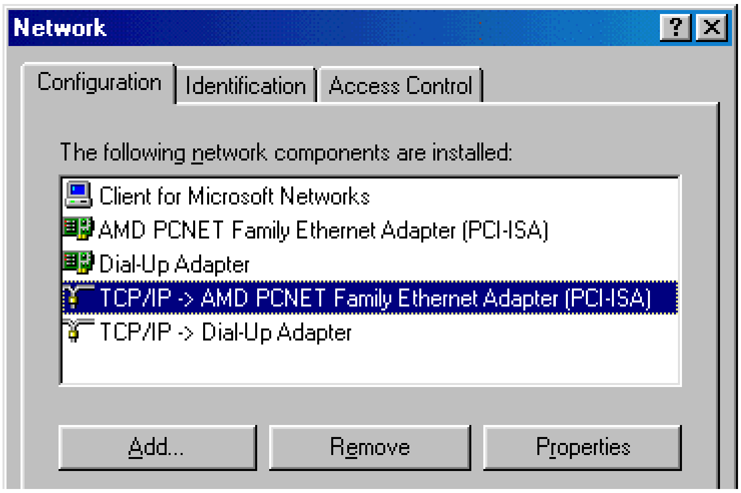
- If an entry for your second Ethernet card appears here, you probably do not need to run any software included with your card, but keep the software handy just in case you need it later to resolve a problem.

4. Note the name of your second Ethernet card.
5. Close the *System Properties* window (the *Control Panel* window should still be open).
6. Open the *Add New Hardware* control panel and follow the on-screen instructions. *We recommend that you allow Windows to search for and install your card automatically*.
7. Restart your computer if Windows gives you the option to do so. Then continue with *Install TCP/IP*.

10.4.2 Install TCP/IP

To determine whether TCP/IP software is already installed on your computer, follow these steps:

1. From the *Start* menu, select *Settings* and then *Control Panel*.
2. Double-click on the *Network* icon. Click on the *Configuration* tab if it is not already selected.



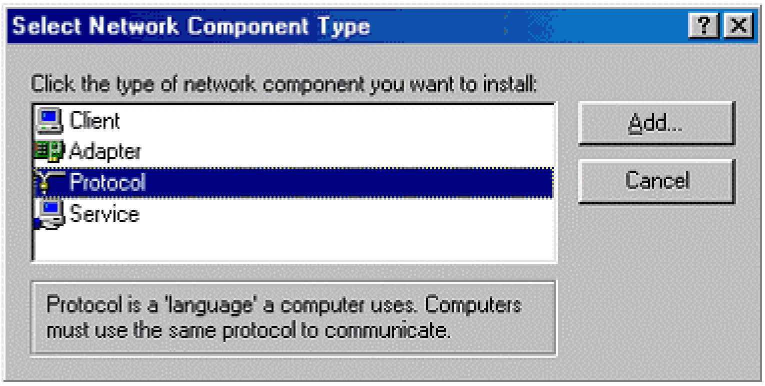
The following network components are installed:

Client for Microsoft Networks
AMD PCNET Family Ethernet Adapter (PCI-ISA)
Dial-Up Adapter
TCP/IP -> AMD PCNET Family Ethernet Adapter (PCI-ISA)
TCP/IP -> Dial-Up Adapter

3. Look in the box labeled *The following network components are installed*.
   a. If you see *IPX/SPX-compatible Protocol* or *NetBEUI* in the list, select it, then click the *Remove* button to delete it. These protocols are used by some networked applications, especially games, but they may interfere with your Ethernet connection.
   b. If you ***don't*** see *TCP/IP* for your second Ethernet card, then continue with step 4.
      If you ***do*** see *TCP/IP* for your second Ethernet card, skip ahead to *Configure TCP/IP*.

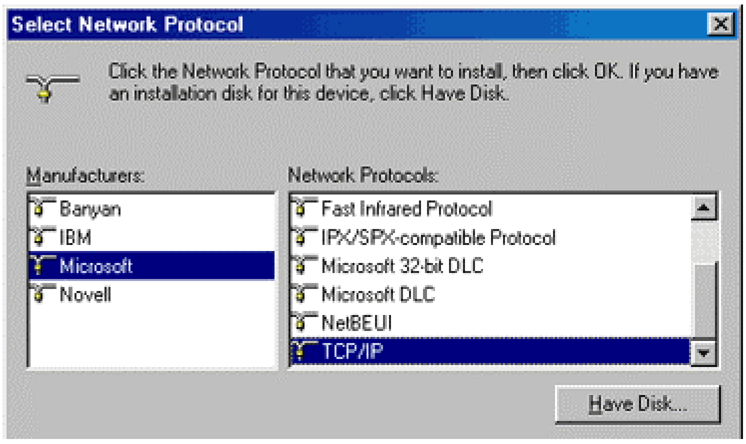
Do these steps only if you do ***not*** *see TCP/IP listed in your Network control panel for your second Ethernet card.*

4. In the *Network* control panel, click the *Add...* button.
5. In the *Select Network Component Type* window, choose *Protocol* and click the *Add...* button.


Select Network Component Type
?
Click the type of network component you want to install:
Client
Adapter
Protocol
Service
Add...
Cancel

Protocol is a 'language' a computer uses. Computers must use the same protocol to communicate.

6. In the *Select Network Protocol* window, select *Microsoft* under *Manufacturer* and *TCP/IP* under *Network Protocols*.



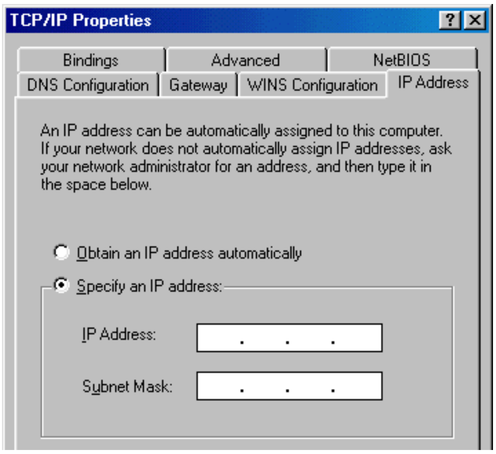
Click the Network Protocol that you want to install, then click OK. If you have an installation disk for this device, click Have Disk.

7. Click the *OK* button to return to the *Network* control panel, then click the *OK* button again to exit the control panel.
8. Restart your computer if Windows gives you the option to do so. Then continue with *Configure TCP/IP*.

10.4.3 Configure TCP/IP

1. From the *Start* menu, select *Settings* and then *Control Panel*. Double-click on the *Network* icon. Click the *Configuration* tab if it is not already selected.
2. In the box labeled *The following network components are installed*, select *TCP/IP*. TCP/IP is listed at least twice, so choose the one followed by the name of your second **Ethernet card** (do *not* choose TCP/IP -> Dial-up Adapter).
3. Click the *Properties* button.
4. In the *TCP/IP Properties* window, click on the *IP Address* tab.

a. Make sure that *Specify an IP address* is selected.
b. Enter `192.168.100.1` for *IP Address* and `255.255.255.0` for *Subnet Mask*.



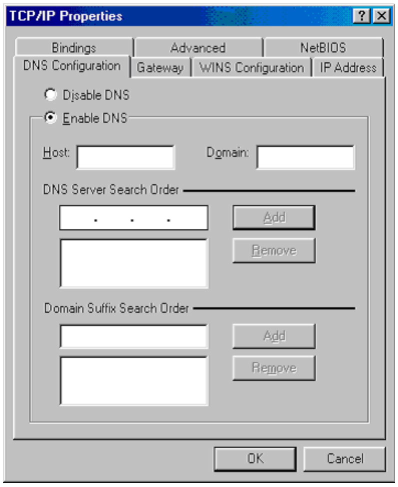
An IP address can be automatically assigned to this computer. If your network does not automatically assign IP addresses, ask your network administrator for an address, and then type it in the space below.

Obtain an IP address automatically [✓ Specify an IP address:]

IP Address: . . .

Subnet Mask: . . .

5. Click on the *DNS Configuration* tab.
. Select *Enable DNS*.
   a. Make sure the *Host* and *Domain* information is blank.



Disable DNS [✓ Enable DNS]

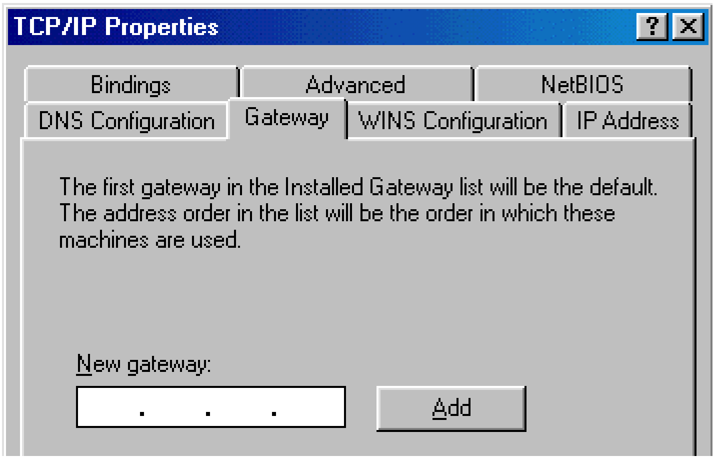
Host: Domain:

DNS Server Search Order

. . .

Domain Suffix Search Order

6. Click on the *Gateway* tab.
   . Make sure the box labeled *New gateway* is blank.



The first gateway in the Installed Gateway list will be the default. The address order in the list will be the order in which these machines are used.

7. Click the *OK* button to return to the *Network* control panel.
8. Click *OK* to exit the *Network* control panel.
9. Restart your computer if Windows gives you the option to do so.

11 Glossary

A

ACB	see Advanced Circular Buffer
A/D (see ADC)	Analog/digital, often used in connection with an A/D converter.
adapter	Alternate designation for a function card that plugs into a backplane, often a PC.
ADC (also see A/D)	Analog-to-Digital Converter. An integrated circuit that converts an analog voltage to a digital number.
ADC conversion	The process of converting an analog input to its digital equivalent.
ADC conversion Start	Signal used to start the process of converting an analog input to a digital value. The source of this signal can be an internal clock or an external asynchronous signal.
ADC Channel List Start	Signal used to start the acquisition of digitized values as defined in the Channel List. The triggering edge of this signal (falling edge) enables the ADC conversion Start signals.
Advanced Circular Buffer	A special user-defined buffer in host memory that stores frames of collected data. The PowerDNA driver allows the user application to fetch data from this buffer in several modes.
alias	A false lower-frequency component that appears in sampled data that has been acquired at an insufficiently high sampling rate.
analog trigger	A trigger that occurs when an analog signal reaches a user-selected level. Users can configure triggering

	to occur at a specific level on either an increasing or a decreasing signal (positive or negative slope).
API	Application Programming Interface, a collection of high-level language function calls that provide access the functions in a driver or other utility.
asynchronous	(1) Hardware—A property of an event that occurs at an arbitrary time, without synchronization to a reference clock.
	(2) Software—A property of a function that begins an operation and returns prior to the completion or termination of the operation.

B

background acquisition	Data is acquired by a DAQ system while another program or processing routine is running without apparent interruption.
base address	A memory address that serves as the starting address for programmable registers. All other addresses are located by adding to the base address.
bipolar	A signal range that includes both positive and negative values (for example, -5V to +5V, also represented as ±5V).
bit	One binary digit, either 0 or 1.
Block mode	A high-speed data transfer in which the address of the data is sent followed by a specified number of back-to-back data words.
Burst mode	A high-speed data transfer in which the address of the data is sent followed by back-to-back data words while a physical signal is asserted.
bus	The group of conductors that interconnect individual circuitry in a computer. Typically, a bus

	is the expansion vehicle to which I/O or other devices are connected. Examples of PC buses are the PCI bus and the PXI bus.
bus master	A type of plug-in board or controller that can read and write to devices on the computer bus without the assistance of the host CPU.
byte	Eight related bits of data, an 8-bit binary number. Also used to denote the amount of memory required to store one byte of data.
C	
cache	High-speed processor memory that buffers commonly used instructions or data to increase processing throughput.
calibration	The setting or correcting of a measuring device or base level, usually by adjusting it to match or conform to a dependably known and unvarying measure.
channel list	For AO Series boards, a set of entries, one for every channel that should be updated. When the simultaneous-update feature is enabled, all channels are usually updated upon a write to the first or last channel in the channel list.
Channel List FIFO	The on-board memory that holds the Channel List.
CL clock	The Channel List clock, also known as the Burst clock, tells the control logic how quickly to move to the next entry in the Channel List and set up the front-end operating parameters such as gain.
control register	Register containing control bits that set up and configure various onboard subsystems.
CMRR	Common-Mode Rejection Ratio, a measure of an instrument's ability to reject interference from a

	common-mode signal, usually expressed in decibels (dB).
code generator	A software program, controlled from an intuitive user interface, that creates syntactically correct high-level source code in languages such as C or Basic.
cold-junction compensation	The means to compensate for the ambient temperature in a thermocouple measurement circuit.
common-mode range	The input range over which a circuit can handle a common-mode signal.
common-mode signal	The mathematical average voltage, relative to the computer's ground, of the signals going into a differential input.
component software	An application that contains one or more component objects that can freely interact with other component software. Examples
	include OLE-enabled applications such as Microsoft Visual Basic and OLE Controls.
conversion time	The time, in an analog input or output system, from the moment a channel is interrogated (such as with a Read instruction) to the moment that accurate data is available.
counter/timer	A circuit that counts external pulses or clock pulses (timing), such as the Intel 8254 device.
coupling	The manner in which a signal is connected from one location to another.
crosstalk	An unwanted signal on one channel due to an input on a different channel.
current drive capability	The amount of current a digital or analog output channel can source or sink while still operating within voltage range specifications.

current sinking	The ability of a DAQ card to dissipate power from an output signal, either analog or digital. Some sensors apply a voltage to a loop, and the DAQ card must be able to accept the resulting current flow.
current sourcing	The ability of a DAQ card to supply current for analog or digital output signals.
CV clock	The Conversion Clock, also known as the Pacer clock, it triggers individual acquisitions and thus tells the A/D how fast to digitize successive samples.

D

D/A	Digital-to-analog, digital/analog
DAC	Digital-to-Analog Converter, an integrated circuit that converts a digital value into a corresponding analog voltage or current.
DAC conversion Start	Signal used to start the process of converting a digital value to an analog output. The source of this signal can be either an internal synchronous clock or an external asynchronous signal.
DAQ	Data Acquisition: (1) Collecting and measuring electrical signals from sensors, transducers, and test probes or fixtures, and moving them to a computer for processing; (2) Collecting and measuring the same kinds of electrical signals with A/D or DIO boards plugged into a PC, and possibly generating control signals with D/A or DIO boards in the same PC.

dB	Decibel, the unit for expressing a logarithmic measure of the ratio of two signal levels: dB = $20\log_{10}(V1/V2)$ for signals in volts.
differential input	An analog-input configuration that measures the difference between signals on two terminals, both of which are isolated from computer ground.
DIO	Digital input/output.
DLL	Dynamic Link Library, a software module in Microsoft Windows containing executable code and data that can be called or used by Windows applications or other DLLs. Functions and data in a DLL are loaded and linked at run time when they are referenced by a Windows application or other DLLs.
DNL	Differential nonlinearity, a measure in LSBs of the worst-case deviation of code widths from their ideal value of 1 LSB.
DMA	Direct Memory Access, a method of transferring data to/from computer memory from/to a device or memory on the bus, taking place while the host processor does something else. DMA is the fastest method of transferring data to/from computer memory.
drivers	Software that controls a specific hardware device such as a DAQ board.
DSP	Digital signal processing.
dual-access memory	Memory that can be sequentially accessed by more than one controller or processor but not simultaneously. Also known as shared memory.
dual-port memory	Memory that can be simultaneously accessed by more than one controller or processor.

dynamic range	The ratio, normally expressed in dB, of the largest signal level in a circuit to the smallest signal level. In DAQ cards it typically refers to the range of signals a board can handle or the amount of noise it suppresses.
E	
EEPROM	Electrically Erasable Programmable Read-Only Memory, a nonvolatile memory device you can repeatedly program for storage, erase and reprogram.
encoder	A device that converts linear or rotary displacement into digital or pulse signals. The most popular type of encoder is the optical encoder.
EPROM	Erasable Programmable Read-Only Memory: A nonvolatile memory device that can be erased (usually by ultraviolet light exposure) and reprogrammed.
ESSI	All DSP56300 devices contain two independent and identical Enhanced Synchronous Serial Interfaces, ESSI0 and ESSI1. Its maximum frequency is the speed of the DSP core divided by four, and thus on most PowerDAQ cards 16.5 MHz.
event	A signal or interrupt generated by a device to notify another device of an asynchronous event. The contents of events are device-dependent.
event-based mode	A board operating mode whereby it notifies the user application of certain predefined subsystem events using Win32 calls. It allows you to write asynchronous applications.
external trigger	A voltage pulse from an external source that triggers an event such as an A/D conversion.

F

FIFO	First-In First-Out, usually used in reference to a memory buffer where the first data stored is the first sent out.
Firmware Simultaneous Update	A method for multichannel updates, when every channel holds its value when new data is written and all channels are updated at the same time when data is written to the specific channel/channels.
fixed point	A format for processing or storing numbers as digital integers. In fixed-point arithmetic all numbers are represented by integers, fractions (usually restricted between ±1.0) or a combination of both integers and fractions. Thus integer mathematics can be implemented on all general-purpose processors.
floating point	Representing data as a combination of a mantissa and an exponent. The mantissa is usually described by a signed fractional value that has a magnitude >= 1.0 and restricted to< 2.0. The exponent, instead, is an integer and represents the number of places any binary number must be shifted, left or right, in order to yield the desired value.
frame	A user-defined number of scans, and these datapoints reside in a predefined portion of a buffer in host-memory. This host-memory buffer is also known as the Advanced Circular Buffer (ACB).
function	A set of software instructions executed by a single line of code that may have input and/or output parameters and returns a value when executed.

G

gain	The factor by which a signal is amplified, sometimes expressed in dB.
gain accuracy	A measure of the deviation of an amplifier's gain from the ideal gain.
GUI	Graphical User Interface, an intuitive means of communicating information to and from a computer program by means of graphical screen displays. GUIs can resemble the front panels of instruments or other objects associated with a computer program.

H

handler	A device driver installed as part of the computer's OS.
hardware	The physical components of a computer system, such as the circuit boards, plug-in boards, chassis, enclosures, peripherals, cables, and so on.
Hardware Simultaneous Update	On AO Series boards, a multichannel update mode whereby when you preprogram the AO logic to update all DACs upon a write to a certain DAC.
High Density Family (HDF)	Applies to AO Series boards; models with 96 D/A outputs.

I

IMD	Intermodulation Distortion, the ratio, in dB, of the total RMS signal level of harmonic sum and difference distortion products, to the overall RMS

	signal level. The test signal consists of two sinewaves added together.
INL	Integral Nonlinearity, a measure in LSB of the worst-case deviation from the ideal A/D or D/A transfer characteristic of the analog I/O circuitry.
input bias current	The current that flows into the inputs of a circuit.
input impedance	The measured resistance and impedance between the input terminals of a circuit.
input offset current	The difference in the input bias currents of the two inputs of an instrumentation amplifier.
instrumentation amplifier	A circuit whose output voltage with respect to ground is proportional to the difference between the voltages at its two inputs.
integral control	A control action that eliminates the offset inherent in proportional control.
integrating A/D	An A/D whose output code represents the average value of the input voltage over a given time interval.
interrupt	A computer signal indicating that the CPU should suspend its current task to service a designated activity.
I/O	Input/Output, the transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data-acquisition and control interfaces.
IPC	Interprocess Communication, protocol by which processes can pass messages. Messages can be either blocks of data and information packets, or instructions and requests for process(es) to perform actions. A process can send messages to itself, other processes on the same machine, or processes located anywhere on the network.

isolation voltage	The voltage that an isolated circuit can normally withstand, usually specified from input to input and/or from any input to the amplifier output, or to the computer bus.
K	
k	kilo, the standard metric prefix for 1000 or 10^3, used with units of measure such as volts, Hertz, and meters.
L	
linearity	The adherence of device response to the equation R = KS, where R = response, S = stimulus, and K is a constant.
LSB	Least-significant bit.
M	
M	mega, the standard metric prefix for 1 million or 10^6, when used with units of measure such as volts and Hertz; the prefix for 1,048,576, or 2^{20}, when used to quantify data or computer memory.
Mbytes/s	A unit for data transfer that means 1 million or 10^6 bytes/sec.
MMI	Man-machine interface, the means by which an operator interacts with an industrial automation system; often called a GUI.
multiplexer	A switching device with multiple inputs that sequentially connects each of its inputs to its output, typically at high speeds, in order to measure several signals with a single analog input channel.

multitasking	A property of an operating system in which several processes can run simultaneously.
mux	see multiplexer

N

noise	An undesirable electrical signal. Noise comes from external sources such as the AC power line, motors, generators, transformers, fluorescent lights, soldering irons, CRT displays, computers, electrical storms, welders, radio transmitters as well as internal sources such as semiconductors, resistors and capacitors.

O

OLE	Object Linking and Embedding, a set of system services that provides a means for applications to interact and interoperate. Based on the underlying Component Object Model, OLE is object-enabling system software. Through OLE Automation, an application can dynamically identify and use the services of other applications. OLE also makes it possible to create compound documents consisting of multiple sources of information from different applications.
OLE controls	see ActiveX controls.
operating system	Base-level software that controls a computer, runs programs, interacts with users, and communicates with installed hardware or peripheral devices.
optical isolation	The technique of using an optoelectronic transmitter and receiver to transfer data without electrical

	continuity to eliminate high potential differences and transients.
OS	see operating system
output settling time	The amount of time required for the analog output voltage of an amplifier to reach its final value within specified limits.
output slew rate	The rate of change of an analog output voltage from one level to another.
overhead	The amount of computer processing resources, such as time or memory, required to accomplish a task.
P	
paging	A technique used for extending the address range of a device to point into a larger address space
PCI	Peripheral Component Interconnect, an expansion bus architecture originally developed by Intel to replace ISA and EISA. It offers a theoretical maximum transfer rate of 132M bytes/sec.
PDXI	PowerDAQ eXtensions for Instrumentation, UEI’s implementation of the PXI bus standard.
PGA	see Programmable-gain amplifier
PID control	A 3-term control algorithm combining proportional, integral and derivative control actions.
pipeline	A high-performance processor structure in which the completion of an instruction is broken into its elements so that several elements can be processed simultaneously from different instructions.
PLC	Programmable logic controller, a special-purpose computer used in industrial monitoring and control applications. PLCs typically have proprietary

	programming and networking protocols and special-purpose digital and analog I/O ports.
Polled mode	DAQ card operating mode whereby the user application queries the board about the status of various subsystems as needed.
port	A communications connection on a computer or a remote controller.
postriggering	The technique used on a DAQ card to acquire a programmed number of samples after trigger conditions are met.
potentiometer	An electrical device whose resistance you can manually adjusted; known among engineers as a “pot.”
pretriggering	The technique used on a DAQ card to keep a continuous buffer filled with data, so that when the trigger conditions are met, the sample includes the data leading up to the trigger condition.
programmable-gain amplifier	also see PGA, an amplifier where you can change the amount of gain applied to the inputs. Gain settings today are usually made with software instead of setting jumpers as was necessary with first-generation DAQ boards.
programmed I/O	The standard method a CPU uses to access an I/O device—each byte of data is read or written by the CPU.
propagation delay	The amount of time required for a signal to pass through a circuit.
proportional control	A control action whose output is proportional to the deviation of the controlled variable from a desired setpoint.

protocol	The exact sequence of bits, characters and control codes used to transfer data between computers and peripherals through a communications channel.
pseudodifferential	An analog-input configuration where all channels refer their inputs to a common ground—but this ground is not connected to the computer ground.
PXI	PCI eXtensions for Instrumentation, a bus standard that combines the mechanical form factor of the CompactPCI specification and the electrical aspects of the PCI bus. It also adds integrated timing and triggering designed specifically for measurement and automation applications.
Q	
quantization error	The inherent uncertainty in digitizing an analog value due to the finite resolution of the conversion process.
R	
real time	A system in which the desired action takes place immediately when all input conditions are fulfilled; it never has to wait for other processes to complete before it can start. In DAQ terms, it generally refers to the processing of data as it is acquired instead of being accumulated and getting processed at a later time.
relative accuracy	A measure in LSB of the accuracy of an A/D. It includes all nonlinearity and quantization errors. It does not include offset and gain errors of the circuitry feeding the ADC.
resolution	The smallest signal increment that a measurement system can detect. Resolution can be expressed in bits, in proportions, or in percent of full scale. For

	example, a system has a resolution equal to 12 bits = one part in 4,096 = 0.0244% of full scale.
resource locking	A technique whereby a device is signaled not to use one of its resources, often local memory, while that resource is being used by another device, generally the system bus.
ribbon cable	A flat cable in which conductors are placed side by side.
RMS	Root-mean square, computed by squaring the instantaneous voltage, integrating over the desired time and taking the square root.
RTD	Resistance temperature detectors operate based on the principle that electrical resistance varies with temperature. They generally use pure metal elements, platinum being the most widely specified RTD element type although nickel, copper, and Balco (nickel-iron) alloys are also used. Platinum is popular due to its wide temperature range, accuracy, stability as well as the degree of standardization among manufacturers. RTDs are characterized by a linear positive change in resistance with respect to temperature. They exhibit the most linear signal over temperature of any electronic sensing device
RTSI	Real Time Systems Integration bus, developed by National Instruments, this intercard bus allows you to transfer data and control signals without using the backplane bus.

S

sample	16-bit binary data that should be converted to the voltage
samples/sec	expresses the rate at which a DAQ board digitizes an analog signal.

scan	one run through the presently configured Channel List
SDK	Software developer's kit, a collection of drivers and utilities that allow engineers to write their own application programs.
SE	see single-ended.
self-calibrating	reference to a DAQ board that calibrates its own A/D and D/A circuits with a reference source, sometimes provided internally with a precision D/A converter.
sensor	A device that generates an electrical signal in response to a physical stimulus (such as heat, light, sound, pressure, motion or flow).
Sequential Update mode	Performs multi channel updates where every write to the analog-output channel immediately leads to a change in the output voltage.
S/H	Sample/Hold, a circuit that acquires and stores an analog voltage on a capacitor for a short period of time.
simultaneous sampling	the act of digitizing multiple channels simultaneously, with interchannel skew often being measured in psec.
Simultaneous update mode	On AO Series boards, this mode (also referred to as Update All) all channels previously written to in the Write&Hold mode update their outputs at the same time.
single-ended	a term used to describe an analog-input configuration where you measure each channel with respect to a common analog ground.
Single-Point Update mode	In an AO Series board, performs an independent update of any available DACs.

Slow Bit	a control bit in the analog-input configuration word that instructs the A/D to wait a short while before actually digitizing the input voltage; it gives the input amplifier time to settle, and is very useful when working with very high gains.
SNR	also S/N ratio or Signal/Noise ratio, the ratio of the peak power level to the remaining noise power, expressed in dB.
software trigger	A programmed event that triggers an event such as a data acquisition.
SPDT	Single-pole double-throw, a switch in which one terminal can be connected to one of two other terminals.
SSH	Simultaneous Sample/Hold, see simultaneous sampling
S/s, S/sec	see samples/sec
strain gage	A sensor that converts mechanical motion into an electronic signal. A change in capacitance, inductance or resistance is proportional to the strain experienced by the sensor, but resistance is the most widely used characteristic that varies in proportion to strain.
Standard Density Family (SDF)	Applies to AO Series boards; all models with from 8 to 32 D/A outputs.
subroutine	A set of software instructions executed by a single line of code that may have input and/or output parameters.
subsystem	On PowerDAQ cards, a group of circuits that perform either analog input, analog output, digital input, digital output or counter/timer functions.
successive-approximation	

A/D	An A/D that sequentially compares a series of binary-weighted values with an analog input to produce an output digital word in n steps, where n is the A/D's resolution in bits.
synchronous	A property of a function that begins an operation and returns only when the operation is complete.
system noise	A measure of the amount of noise seen by an analog circuit or an A/D when the analog inputs are grounded.
T	
TCP/IP	Transmission Control Protocol/Internet Protocol, the basic multi-layer communication protocol of the Internet but that is also used in a private network (either an intranet or an extranet). The higher layer, TCP, manages the assembling of a message or file into smaller packets that are transmitted and received by a TCP layer that reassembles the packets into the original message. IP handles the address portion of each packet so it gets to the right destination.
THD	Total harmonic distortion, the ratio of the total RMS signal due to harmonic distortion to the overall RMS signal, expressed in dB or percent.
THD+N	The percentage of Total Harmonic Distortion + Noise (THD+N) of a sine wave equals 100 times the ratio of the RMS voltage measured with the fundamental component of a sine wave removed by a notch filter, to the RMS voltage of the fundamental component.
thermistor	A temperature-sensing element that exhibits a large change in resistance proportional to a small change in temperature. Thermistors usually have negative

	temperature coefficients. They tend to be more accurate than thermocouples or RTDs, but they have a much more limited temperature range.
thermocouple	A temperature sensor created by joining two dissimilar metals. The junction produces a small voltage as a function of temperature.
throughput rate	The flow of data, measured in bytes/sec, for a given continuous operation.
transducer	A device that converts energy from one form to another. Generally applied to devices that convert a physical phenomenon (such as pressure, temperature, humidity, or flow) to an electrical signal.
transfer rate	The rate, measured in bytes/sec, at which data is moved from a source to a destination after software initialization and setup operations; the maximum rate at which the hardware can operate.
Trigger	A signal, in either hardware or software, that initiates or halts a process. In DAQ boards, it generally refers to a signal that starts or stops an A/D, D/A or DIO operation.

U

UCT	User counter/timer
unipolar	A signal range that is always positive (e.g. 0 to 10 V).
Update All	Applicable to AO Series boards; see Simultaneous Update mode

W

Write&Hold mode	On AO Series boards, a mode whereby data is written to the output register but the output voltage

	remains unchanged and stays at the previous update value.

Z

zero offset	The difference between true zero and an indication given by a measuring instrument.
zero-overhead looping	The ability of a high-performance processor to repeat instructions without requiring time to branch to the beginning of the instructions.
zero-wait-state memory	Memory fast enough that the processor does not have to wait during any reads and writes to the memory.

12 Index